	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	7 May, 2001	4 September, 2015	GEN-CXXXXX-REVA	1 of 9

Author: Laurent Lefebvre

Issue To:	**Copy No:**

R400 Sequencer Specification

SEQ

Version 0.1

Overview: This is an architectural specification for the R400 Sequencer block (SEQ). It provides an overview of the required capabilities and expected uses of the block. It also describes the block interfaces, internal sub-blocks, and provides internal state diagrams.

AUTOMATICALLY UPDATED FIELDS:
Document Location: D:\Perforce\r400\arch\doc\gfx\MC\R400 MemCtl.doc
Current Intranet Search Title: R400 Memory Controller Architectural Specification

APPROVALS	
Name/Dept	Signature/Date

Remarks:

THIS DOCUMENT CONTAINS CONFIDENTIAL INFORMATION THAT COULD BE SUBSTANTIALLY DETRIMENTAL TO THE INTEREST OF ATI TECHNOLOGIES INC. THROUGH UNAUTHORIZED USE OR DISCLOSURE.

ATI 2007
LG v. ATI
IPR2015-00326

Table Of Contents

1. OVERVIEW 3
1.1 Top Level Block Diagram 4
2. TEXTURE ARBITRATION 7
3. ALU ARBITRATION 8
4. INPUT INTERFACE 8
4.1 Rasterizer to Register File (interpolated data) 8
4.2 Texture Unit to Register File (texture return) 8
4.3 ALU Unit to Register File (ALU op result) 8
4.4 Scalar Unit to Register File (Scalar op result) 8
5. OUTPUT INTERFACE 8
5.1 Sequencer to Shader Engine Bus 8
5.2 Shader Engine to Texture Unit Bus 9
6. OPEN ISSUES 9

Revision Changes:

Rev 0.1 (Laurent Lefebvre) Date: May 7, 2001	First draft.

1. Overview

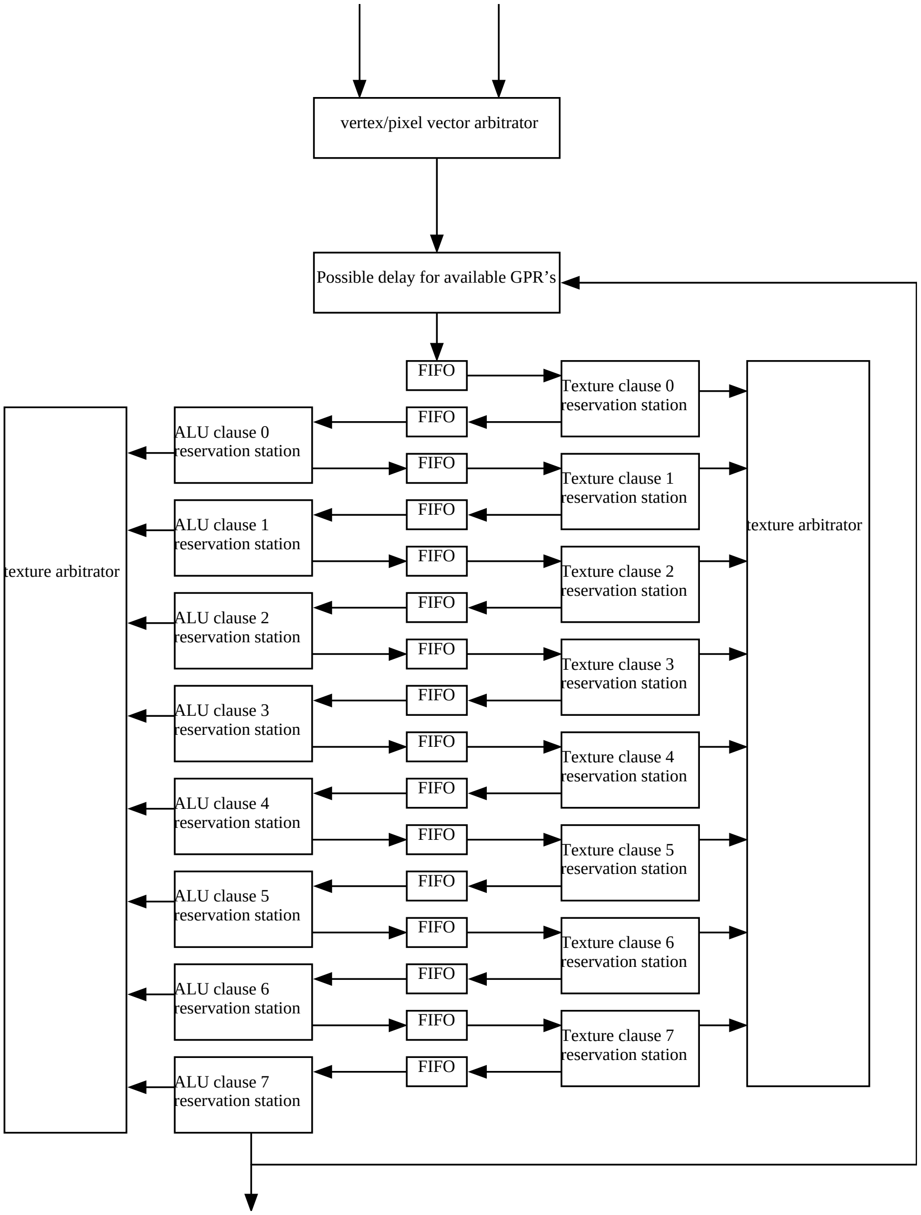
The sequencer first arbitrates between vectors of 16 vertices that arrive directly from primitive assembly and vectors of 8 quads (32 pixels) that are generated in the raster engine.

The vertex or pixel program specifies how many GPR's it needs to execute. The sequencer will not start the next vector until the needed space is available.

The sequencer is based on the R300 design. It chooses an ALU clause and a texture clause to execute, and execute all of the instructions in a clause before looking for a new clause of the same type. Each vector will have eight texture and eight alu clauses, but clauses do not need to contain instructions. A vector of pixels or vertices ping-pongs along the sequencer FIFO, bouncing from texture reservation station to alu reservation station. A FIFO exists between each reservation stage, holding up vectors until the vector currently occupying a reservation station has left. A vector at a reservation station can be chosen to execute. The sequencer looks at all eight alu reservation stations to choose an alu clause to execute and all eight texture stations to choose a texture clause to execute. The arbitrator will give priority to clauses/reservation stations closer to the top of the pipeline. It will not execute an alu clause until the texture fetches initiated by the previous texture clause have completed.

To support the shader pipe the raster engine also contains the shader instruction cache and constant store.

1.1 Top Level Block Diagram


vertex/pixel vector arbitrator
Possible delay for available GPR's
FIFO
Texture clause 0 reservation station
Texture clause 1 reservation station
Texture clause 2 reservation station
Texture clause 3 reservation station
Texture clause 4 reservation station
Texture clause 5 reservation station
Texture clause 6 reservation station
Texture clause 7 reservation station
ALU clause 0 reservation station
ALU clause 1 reservation station
ALU clause 2 reservation station
ALU clause 3 reservation station
ALU clause 4 reservation station
ALU clause 5 reservation station
ALU clause 6 reservation station
ALU clause 7 reservation station
texture arbitrator
texture arbitrator

The rasterizer always checks the vertices FIFO first and if allowed by the sequencer sends the data to the shader. If the vertex FIFO is empty then, the rasterizer takes the first entry of the pixel FIFO (a vector of 32 pixels) and sends it to the interpolators. Then the sequencer takes control of the packet.

On receipt of a packet, the input state machine (not pictured but just before the first FIFO) allocated enough space in the registers to store the interpolatoted values and temporaries. Following this, the input state machine stacks the packet in the first FIFO.

On receipt of a command, the level 0 texture machine issues a texure request and corresponding register address for the texture address (ta). A small command (tcmd) is passed to the texture system identifying the current level number

(0) as well as the register set being used. One texture request is sent every 4 clocks causing the texturing of four 2x2s worth of data.

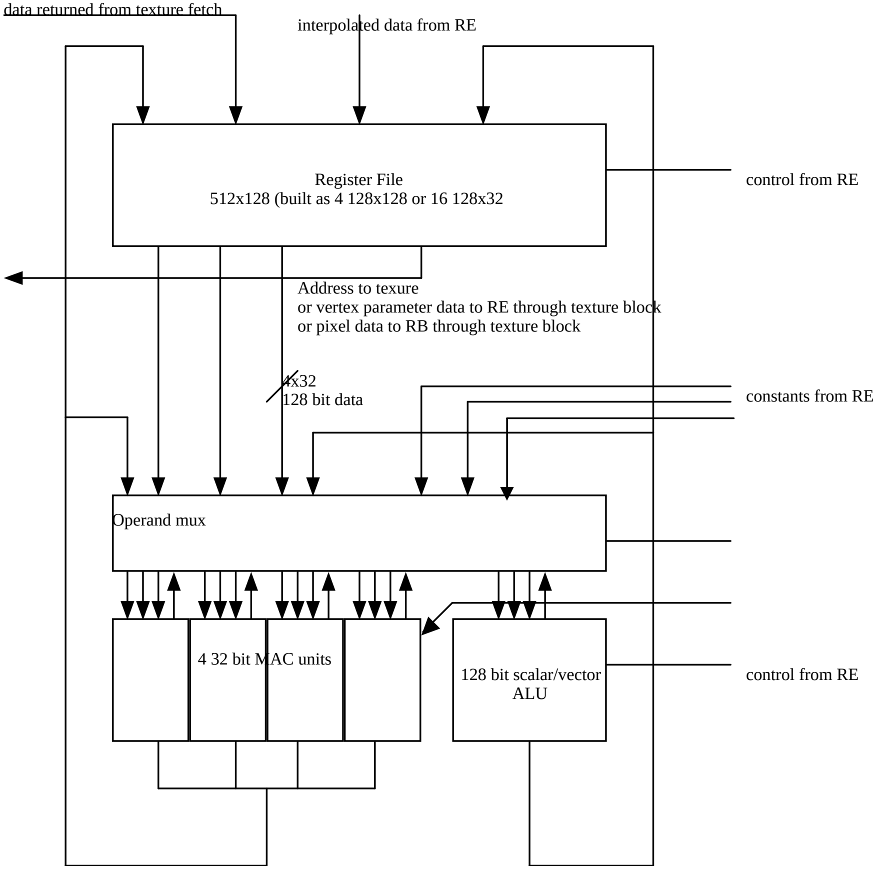
Uppon recept of the return data (identified by the tcmd containing the level number 0), the level 0 texture machine issues a register address for the return value (td). Then, it puts the finished packet in FIFO 1.

On receipt of a command, the level 0 ALU machine issues a complete set of level 0 shader instructions. For each instruction, the state machine generates 3 source addresses, one destination address (2 cycles later) and an instruction id wich is used to index into the instruction store. Once the last instruction as been issued, the packet is put into FIFO 2. Note that in the case of a pixel packet, the two vectors of 16 pixels are interleaved in order to hide the latency of the ALUs (8 cycles).

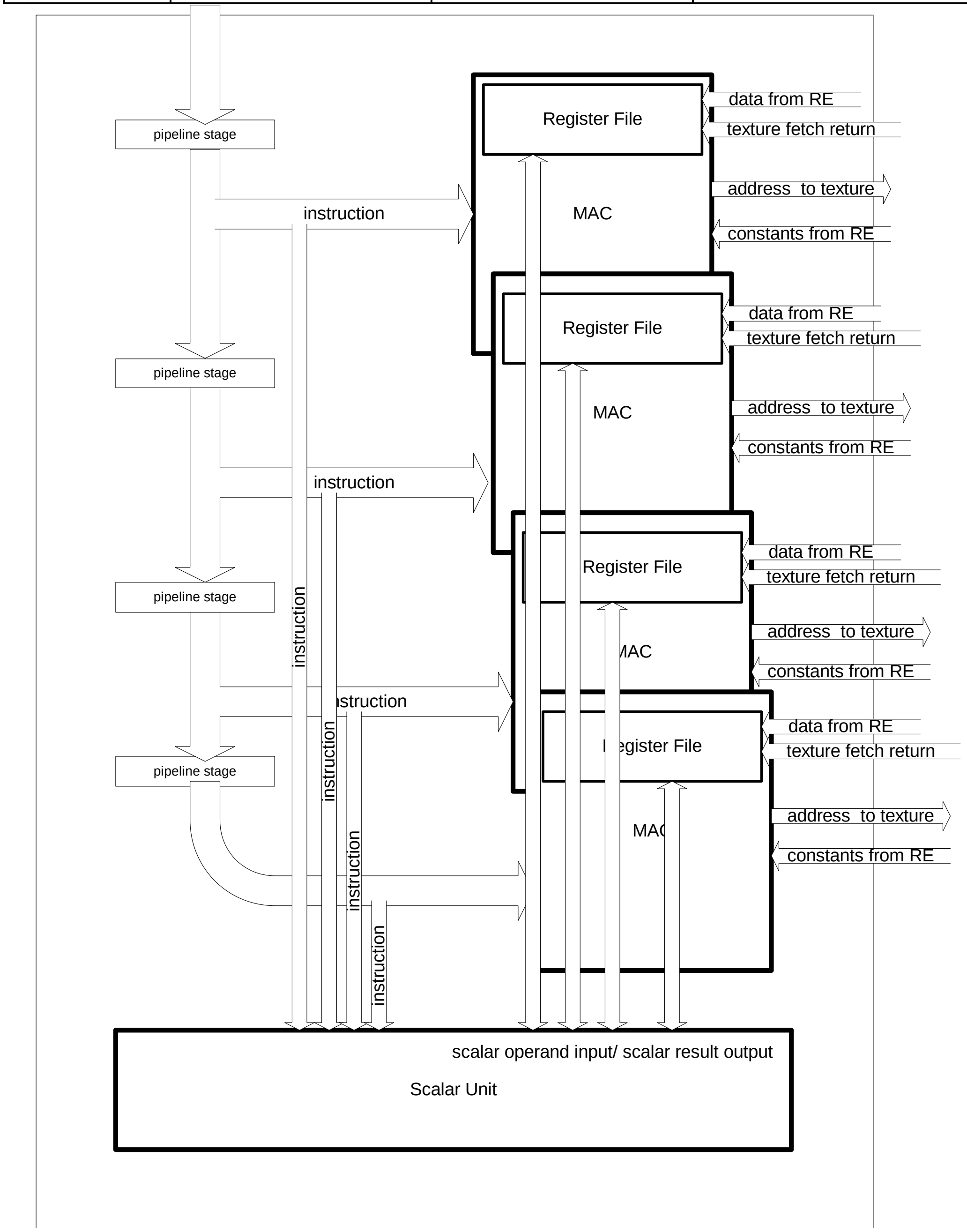
All other level process in the same way until the packet finally reaches the last ALU machine (8). On completion of the level 8 ALU clause, a valid bit is sent to the Render Backend wich picks up the color data. This requires that the last instruction writes to the output register – a condition that is almost always true. If the packet was a vertex packet, instead of sending the valid bit to the RB, it is sent to the PA, which picks up the data a puts it into the vertex store.

Only one ALU state machine may have access to the SRAM address bus or the instruction decode bus at one time. Similarly, only one texture state machine may have access to the SRAM address bus at one time. Arbitration is performed by two arbitrer blocks (one for the ALU state machines and one for the texture state machines). The arbitrers always favor the higher number state machines, preventing a bunch of half finished jobs from clogging up the SRAMS.

Each state machine maintains an address pointer specifying where the 16 (or 32) entries vector is located in the SRAM (the texture machine has two pointers one for the read address and one for the write). Upon completion of its job, the address pointer is incremented by a predefined amount equal to the total number of registers required by the shading code. A comparison of the address pointer for the first state machine in the chain (the input state machine), and the last machine in the chain (the level 8 ALU machine), gives an indication of how much unallocated SRAM memory is available. When this number falls below a preset watermark, the input state machine will stall the rasterizer preventing new data from entering the chain.



data returned from texture fetch
interpolated data from RE
Register File
512x128 (built as 4 128x128 or 16 128x32
control from RE
Address to texure
or vertex parameter data to RE through texture block
or pixel data to RB through texture block
4x32
128 bit data
constants from RE
Operand mux
4 32 bit MAC units
128 bit scalar/vector
ALU
control from RE



2. Texture Arbitration

The texture arbitration logic chooses one of the 8 potentially pending texture clauses to be executed. The choice is made by looking at the fifos from 8 to 0 and picking the first one ready to execute. Once chosen, the clause state machine will send one 2x2 texture fetch per 4 clocks until all the texture fetch instructions of the clause are sent. This means that there cannot be any dependencies between two texture fetches of the same clause.

3. ALU Arbitration

ALU arbitration proceeds in almost the same way than texture arbitration. The ALU arbitration logic chooses one of the 8 potentially pending ALU clauses to be executed. The choice is made by looking at the fifos from 8 to 0 and picking the first one ready to execute. If the packet chosen is a packet of vertices, the state machine issues one instruction every 4 clocks until the clause is finished. This means that the compiler has to insert nops between two dependent successive instructions. If the packet is a pixel packet it is made out of two sub-vectors of 16. Thus the state machine issues the first instruction for the first sub-vector and then, 4 clocks later, the first instruction of the second sub-vector and so on until the clause is finished. Proceeding this way hides the latency of 8 clocks of the ALUs.

4. Input Interface

4.1 Rasterizer to Register File (interpolated data)

Name	Direction	bits	Description
SND	SEQ→SP	1	High when sending data
Interpolated data	SEQ→SP	512	512 bits transferred every 4 cycles

4.2 Texture Unit to Register File (texture return)

Name	Direction	bits	Description
SND	SEQ→TU	1	High when sending data
Texture colors	TU→SP	512	512 bits transferred every 4 cycles

4.3 ALU Unit to Register File (ALU op result)

Name	Direction	bits	Description
SND	SEQ→SP	1	High when sending data
Blend result ALU	SP→SP	512	512 bits transferred every 4 cycles
Write Mask	SP→SP	16	The four write masks

4.4 Scalar Unit to Register File (Scalar op result)

Name	Direction	bits	Description
SND	SEQ→SP	1	High when sending data
Scalar result	SP→SP	512	512 bits transferred every 4 cycles
Write Mask	SP→SP	16	The four write masks

5. Output Interface

5.1 Sequencer to Shader Engine Bus

This is a bus that sends the instruction and constant data to all 4 Sub-Engines of the Shader. Because a new instruction is needed only every 4 clocks, the width of the bus is divided by 4 and both constants and instruction are sent over those 4 clocks.

Name	Direction	Bits	Description
Instruction Start	SEQ-> SP	1	High on first cycle of transfer
Constant 0	SEQ-> SP	32	128 bits transferred over 4 cycles, alpha first…blue last
Constant 1	SEQ-> SP	32	128 bits transferred over 4 cycles, alpha first…blue last
Instruction	SEQ-> SP	40	160 bits transferred over 4 cycles

5.2 Shader Engine to Texture Unit Bus

One quad’s worth of addresses is transferred to Texture Unit every clock. These are sourced fro a different pixel within each of the sub-engines repeating every 4 clocks. The register file index to read must precede the data by 2 clocks. The Read address associated with Quad 0 must be sent 1 clock after the Instruction Start signal is sent, so that data is read 3 clocks after the Instruction Start.

One Quad’s worth of Texture Data may be written to the Register File every clock. These are directed to a different pixel of the sub-engines repeating every 4 clocks. The register file index to write must accompany the data. Data and Index associated with the Quad 0 must be sent 3 clocks after the Instruction Start signal is sent.

Name	Direction	Bits	Description
Tex_Read_Register_Index	SEQ->SP	8	Index into Register Files for reading Texture Address
Tex_RegFile_Read_Data	SP->TEX	512	4 Texture Addresses read from the Register File
Tex_Write_Register_Index	SEQ->SP	8	Index into Register file for write of returned Texture Data

6. Open issues

There is currently an issue with constants. If the constants are not the same for the whole vector of vertices, we don’t have the bandwith from the texture store to feed the ALUs. Two solutions exists for this problem:

1) Let the compiler handle the case and put those instructions in a texture clause so we can use the bandwith there to operate. This requires a significant amount of temporary storeage in the register store.
2) Waterfall down the pipe allowing only at a given time the vertices having the same constants to operate in parralel. This might in the worst case slow us down by a factor of 16.

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	7 May, 2001	8 September, 2015~~3~~ September, 201512	GEN-CXXXXX-REVA	1 of 16

Author: Laurent Lefebvre

Issue To:	**Copy No:**

R400 Sequencer Specification

SEQ

Version 0.3~~2~~

Overview: This is an architectural specification for the R400 Sequencer block (SEQ). It provides an overview of the required capabilities and expected uses of the block. It also describes the block interfaces, internal sub-blocks, and provides internal state diagrams.

AUTOMATICALLY UPDATED FIELDS:
Document Location: C:\perforce\r400\arch\doc\gfx\RE\R400_Sequencer.doc
Current Intranet Search Title: R400 Sequencer Specification

APPROVALS	
Name/Dept	Signature/Date

Remarks:

THIS DOCUMENT CONTAINS CONFIDENTIAL INFORMATION THAT COULD BE SUBSTANTIALLY DETRIMENTAL TO THE INTEREST OF ATI TECHNOLOGIES INC. THROUGH UNAUTHORIZED USE OR DISCLOSURE.

Table Of Contents

1. OVERVIEW... 3
1.1 Top Level Block Diagram 4
1.2 Data Flow graph................................ 8~~7~~
1.3 Control Graph.................................. 11~~10~~
2. INTERPOLATED DATA BUS....... 11~~10~~
3. INSTRUCTION STORE .... 11~~10~~
4. CONSTANT STORE..................... 12~~11~~
5. LOOPING AND BRANCHES........ 12~~11~~
6. REGISTER FILE ALLOCATION... 12~~11~~
7. TEXTURE ARBITRATION............ 13~~12~~
8. ALU ARBITRATION .... 13~~12~~
9. HANDLING STALLS .... 14~~13~~
10. CONTENT OF THE RESERVATION STATION FIFOS.................................... 14~~13~~
11. THE OUTPUT FILE (RB FIFO AND PARAMETER CACHE).......................... 14~~13~~
12. INTERFACES........................ 14~~13~~
12.1 External Interfaces............. 14~~13~~
12.1.1 Sequencer to Shader Engine Bus....................................... 14~~13~~
12.1.2 Shader Engine to Output File 14~~13~~
12.1.3 Shader Engine to Texture Unit Bus (Fast Bus) .... 15~~14~~
12.1.4 Sequencer to Texture Unit bus (Slow Bus) 15~~14~~
12.1.5 Shader Engine to RE/PA Bus 15~~14~~
12.1.6 PA to sequencer................ 15~~14~~
13. OPEN ISSUES............................. 16~~14~~
~~**1. OVERVIEW... 3**~~
~~1.1 Top Level Block Diagram .... 4~~
~~**2. TEXTURE ARBITRATION.................. 9**~~
~~**3. ALU ARBITRATION .... 9**~~
~~**4. HANDLING STALLS .... 10**~~
~~**5. CONTENT OF THE RESERVATION STATION FIFOS.. 10**~~
~~**6. INTERFACES............................. 10**~~
~~6.1 External Interfaces.................... 10~~
~~6.1.1 Sequencer to Shader Engine Bus 10~~
~~6.1.2 Shader Engine to Output File 10~~
~~6.1.3 Shader Engine to Texture Unit Bus (Fast Bus).. 10~~
~~6.1.4 Sequencer to Texture Unit bus (Slow Bus).. 11~~
~~6.1.5 Shader Engine to RE/PA Bus... 11~~
~~**7. OPEN ISSUES.................................. 11**~~

Revision Changes:

Rev 0.1 (Laurent Lefebvre) Date: May 7, 2001	First draft.
Rev 0.2 (Laurent Lefebvre) Date : July 9, 2001	Changed the interfaces to reflect the changes in the SP. Added some details in the arbitration section.
Rev 0.3 (Laurent Lefebvre) Date : August 6, 2001	Reviewed the Sequencer spec after the meeting on August 3, 2001.

1. Overview

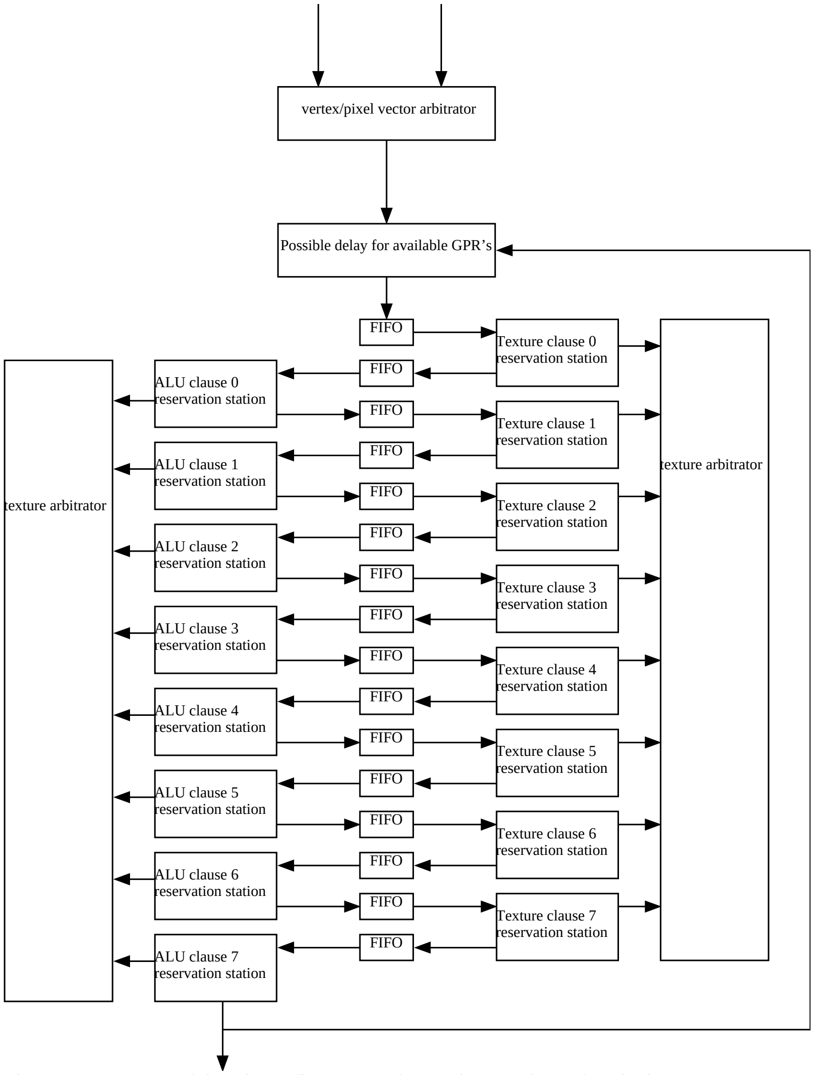
The sequencer first arbitrates between vectors of 16 ~~(maybe 32)~~ vertices that arrive directly from primitive assembly and vectors of ~~8~~4 quads (16 pixels) ~~(32 pixels)~~ that are generated in the raster engine.

The vertex or pixel program specifies how many GPR's it needs to execute. The sequencer will not start the next vector until the needed space is available.

The sequencer is based on the R300 design. It chooses ~~an~~ two ALU clauses and a texture clause to execute, and executes all of the instructions in ~~a~~a clause before looking for a new clause of the same type. Two ALU clauses are executed interleaved to hide the ALU latency. Each vector will have eight texture and eight ALU clauses, but clauses do not need to contain instructions. A vector of pixels or vertices ping-pongs along the sequencer FIFO, bouncing from texture reservation station to alu reservation station. A FIFO exists between each reservation stage, holding up vectors until the vector currently occupying a reservation station has left. A vector at a reservation station can be chosen to execute. The sequencer looks at all eight alu reservation stations to choose an alu clause to execute and all eight texture stations to choose a texture clause to execute. The arbitrator will give priority to clauses/reservation stations closer to the ~~top~~ bottom of the pipeline. It will not execute an alu clause until the texture fetches initiated by the previous texture clause have completed. There are two separate sets of reservation stations, one for pixel vectors and one for vertices vectors. This way a pixel can pass a vertex and a vertex can pass a pixel.

To support the shader pipe the raster engine also contains the shader instruction cache and constant store. There are only one constant store for the whole chip and one instruction store. These will be shared among the four shader pipes.

1.1 Top Level Block Diagram


vertex/pixel vector arbitrator
Possible delay for available GPR's
FIFO
Texture clause 0 reservation station
Texture clause 1 reservation station
Texture clause 2 reservation station
Texture clause 3 reservation station
Texture clause 4 reservation station
Texture clause 5 reservation station
Texture clause 6 reservation station
Texture clause 7 reservation station
ALU clause 0 reservation station
ALU clause 1 reservation station
ALU clause 2 reservation station
ALU clause 3 reservation station
ALU clause 4 reservation station
ALU clause 5 reservation station
ALU clause 6 reservation station
ALU clause 7 reservation station
texture arbitrator
texture arbitrator

There are two sets of the above figure, one for vertices and one for pixels.

The rasterizer always checks the vertices FIFO first and if allowed by the sequencer sends the data to the shader. If the vertex FIFO is empty then, the rasterizer takes the first entry of the pixel FIFO (a vector of ~~32~~ 16 pixels) and sends it to the interpolators. Then the sequencer takes control of the packet. The packet consists of 3 bits of state, 6-7 bits for the base address of the Shader program and some information on the coverage to determine texture LOD. All other information (2x2 adresses) is put in a FIFO (one for the pixels and one for the vertices) and retrieved when the packet finishes its last clause.

On receipt of a packet, the input state machine (not pictured but just before the first FIFO) allocated enough space in the registers to store the interpolated values and temporaries. Following this, the input state machine stacks the packet in the first FIFO.

On receipt of a command, the level 0 texture machine issues a texure request and corresponding register address for the texture address (ta). A small command (tcmd) is passed to the texture system identifying the current level number (0) as well as the register ~~set being used~~write address for the texture return data. One texture request is sent every 4 clocks causing the texturing of four 2x2s worth of data (or 16 vertices). Once all the requests are sent the packet is put in FIFO 1.

Upon recept of the return data (identified by the tcmd containing the level number 0), the level 0 texture machine issues a register address for the return value (td). Then, it increments the counter of FIFO ~~one~~ 1 to signify to the ALU 1 that the data is ready to be processed.

On receipt of a command, the level 0 ALU machine first decrements the input FIFO counter and then issues a complete set of level 0 shader instructions. For each instruction, the state machine generates 3 source addresses, one destination address (~~2~~ 3 cycles later) and an instruction~~ id wich is used to index into the instruction store~~. Once the last instruction as been issued, the packet is put into FIFO 2.~~ Note that in the case of a pixel packet, the two vectors of 16 pixels are consecutive in order to hide the latency of the ALUs (8 cycles).~~

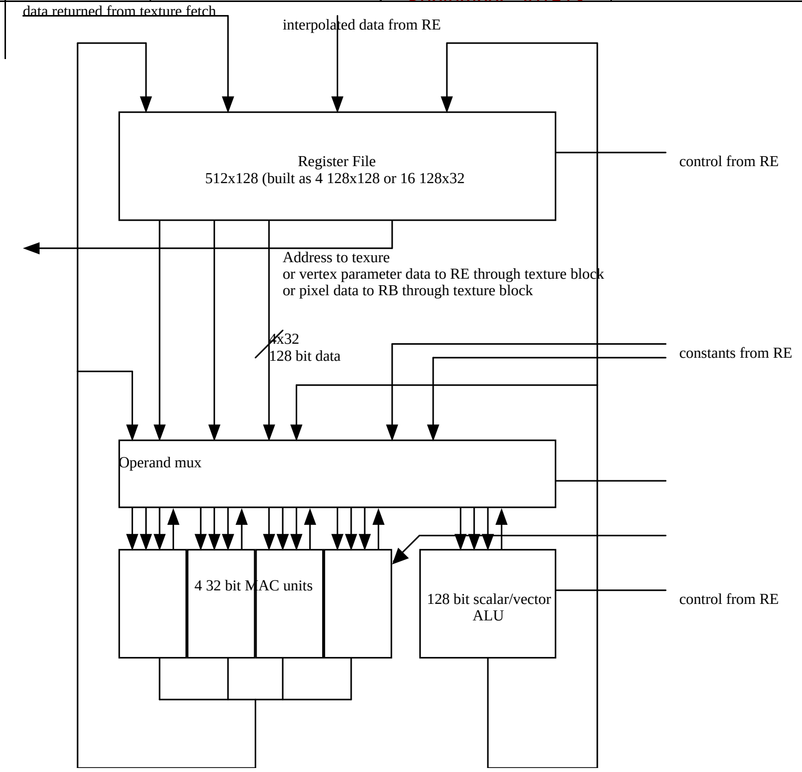
There will always be two active ALU clauses at any given time (and two arbitrers)~~ In this case, the instructions of a vector are interleaved with the instructions of the other vector~~. One arbitrer will arbitrate over the odd clock cycles and the other one will arbitrate over the even clock cycles. The only constraints between the two arbitrers is that they are not allowed to pick the same clause number as they other one is currently working on if the packet os of the same type.

If the packet is a vertex packet, upon reaching ALU clause 4, it can export the position if the position is ready. So the arbitrer must prevent ALU clause 4 to be selected if the positional buffer is full (or can't be accessed). Along with the positional data, the location where the vertex data is to be put is also sent (parameter data pointers).

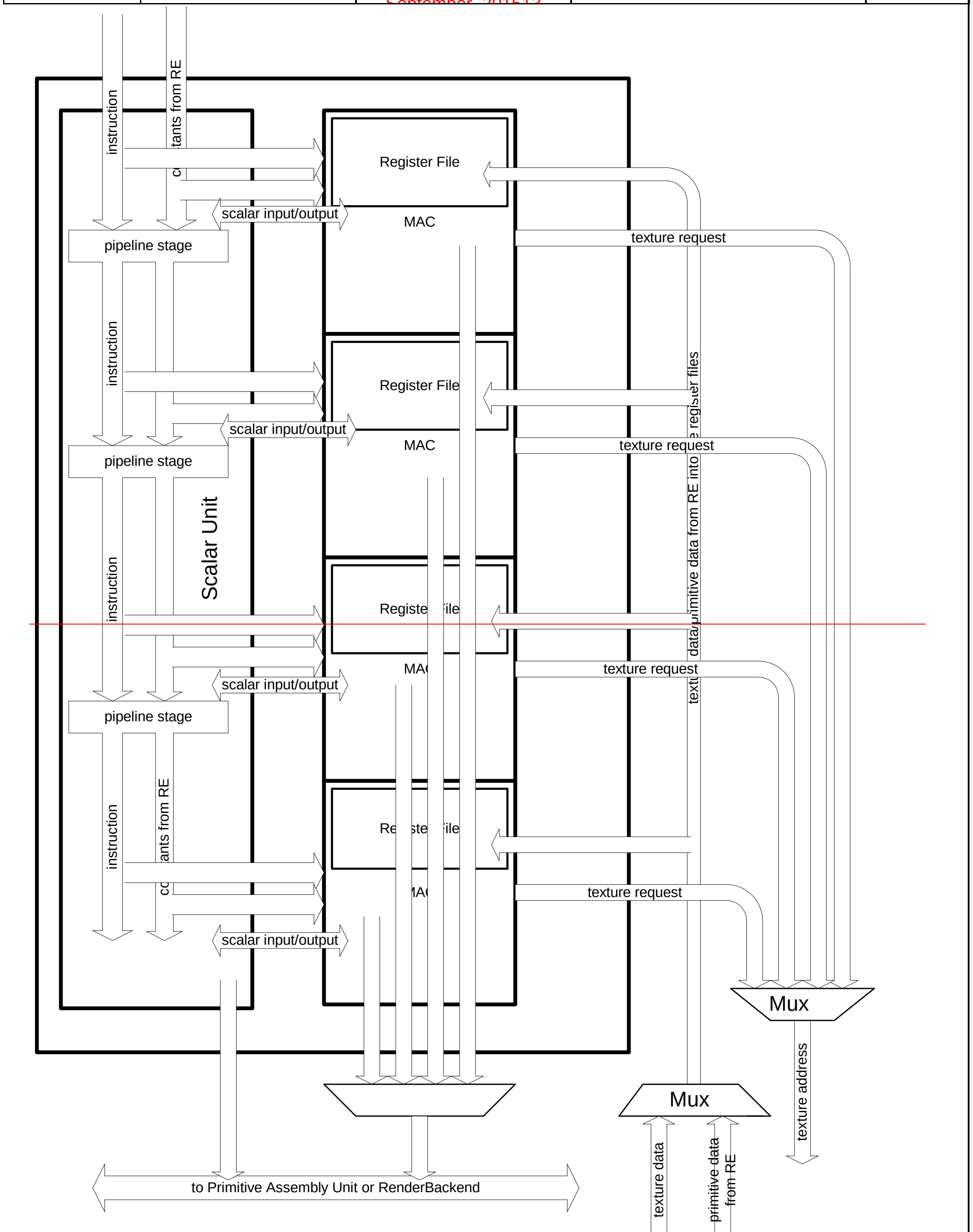
All other level process in the same way until the packet finally reaches the last ALU machine (8). On completion of the level 8 ALU clause, a valid bit is sent to the Render Backend which picks up the color data. This requires that the last instruction writes to the output register – a condition that is almost always true. If the packet was a vertex packet, instead of sending the valid bit to the RB, it is sent to the PA~~, which picks up the data and puts it into the vertex store~~ so it can know that the data present in the parameter store is valid.

Only ~~one~~ two ALU state machine may have access to the ~~SRAM~~register file address bus or the instruction decode bus at one time. Similarly, only one texture state machine may have access to the ~~SRAM~~register file address bus at one time. Arbitration is performed by ~~two~~ three arbitrer blocks (~~one~~ two for the ALU state machines and one for the texture state machines). The arbitrers always favor the higher number state machines, preventing a bunch of half finished jobs from clogging up the ~~SRAM~~register ~~S~~files.

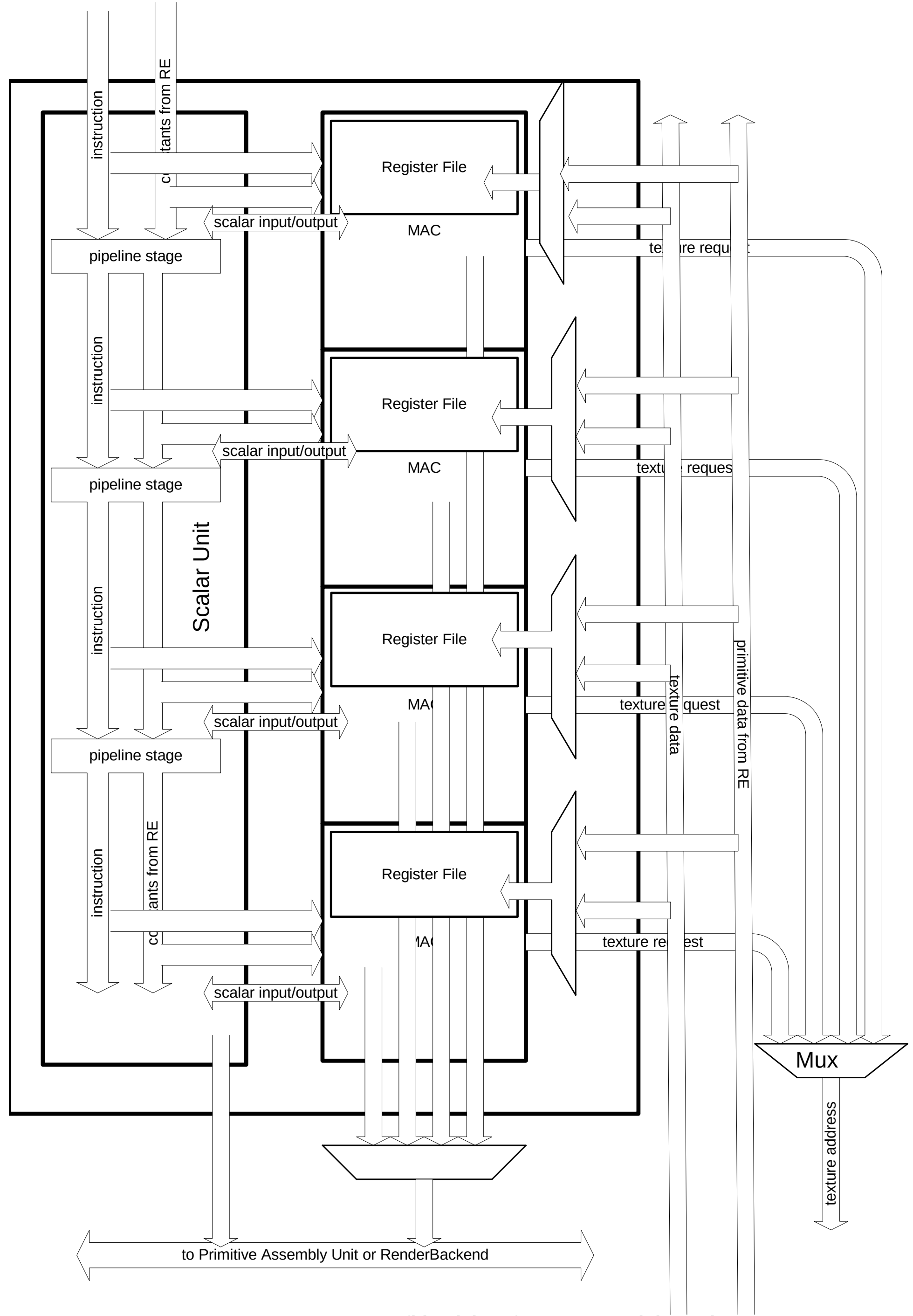
Each state machine maintains an address pointer specifying where the 16~~ (or 32)~~ entries vector is located in the ~~SRAM~~register file (the texture machine has two pointers one for the read address and one for the write). Upon completion of its job, the address pointer is incremented by a predefined amount equal to the total number of registers required by the shading code. A comparison of the address pointer for the first state machine in the chain (the input state machine), and the last machine in the chain (the level 8 ALU machine), gives an indication of how much unallocated ~~SRAM~~register file memory is available

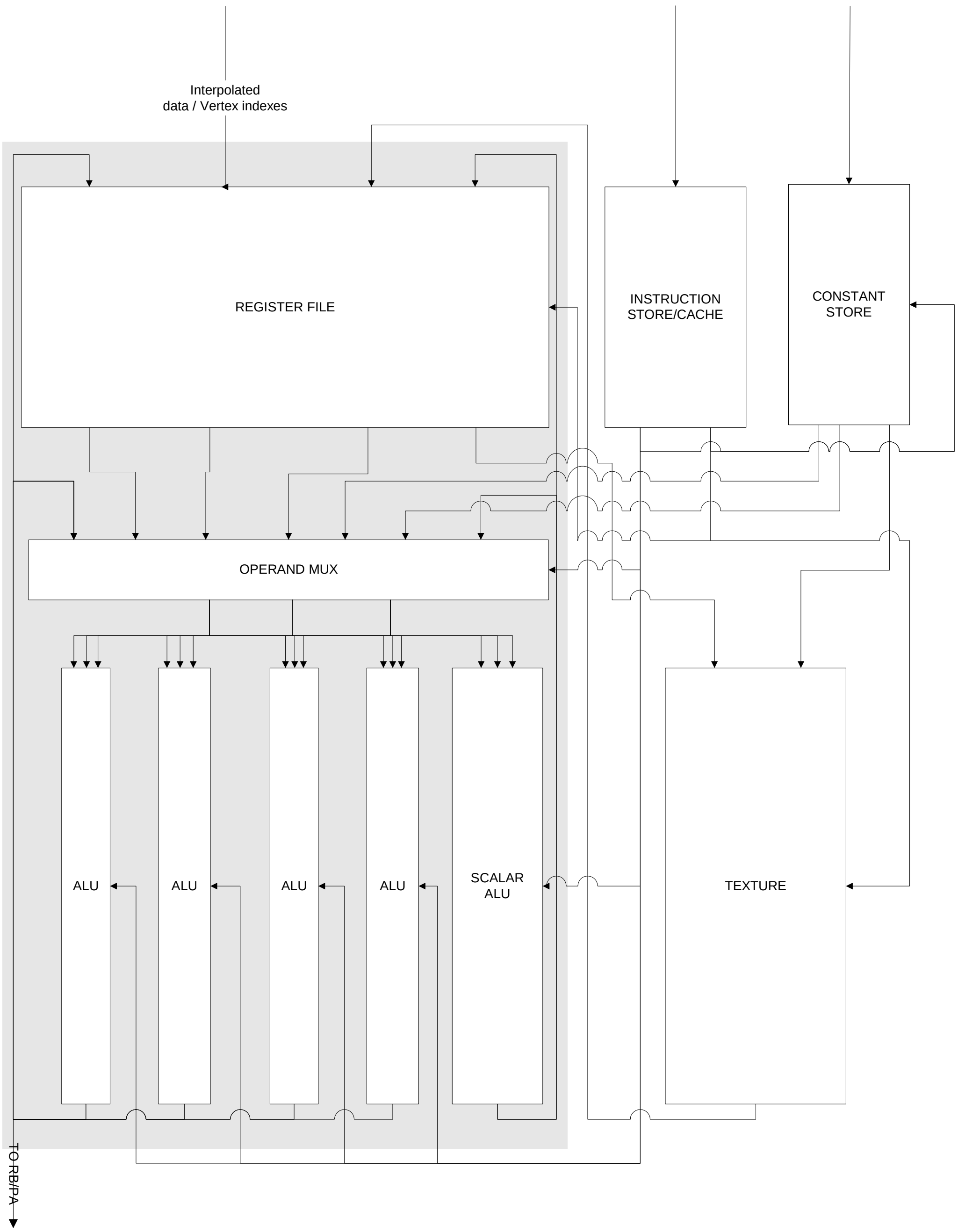


data returned from texture fetch
interpolated data from RE
Register File
512x128 (built as 4 128x128 or 16 128x32
control from RE
Address to texure
or vertex parameter data to RE through texture block
or pixel data to RB through texture block
4x32
128 bit data
constants from RE
Operand mux
4 32 bit MAC units
128 bit scalar/vector ALU
control from RE


instruction
constants from RE
Register File
MAC
scalar input/output
pipeline stage
texture request
instruction
Register File
MAC
scalar input/output
pipeline stage
Scalar Unit
texture data/primitive data from RE into the register files
instruction
Register File
MAC
scalar input/output
texture request
pipeline stage
texture request
instruction
constants from RE
Register File
MAC
texture request
scalar input/output
Mux
texture address
Mux
texture data
primitive data from RE
to Primitive Assembly Unit or RenderBackend

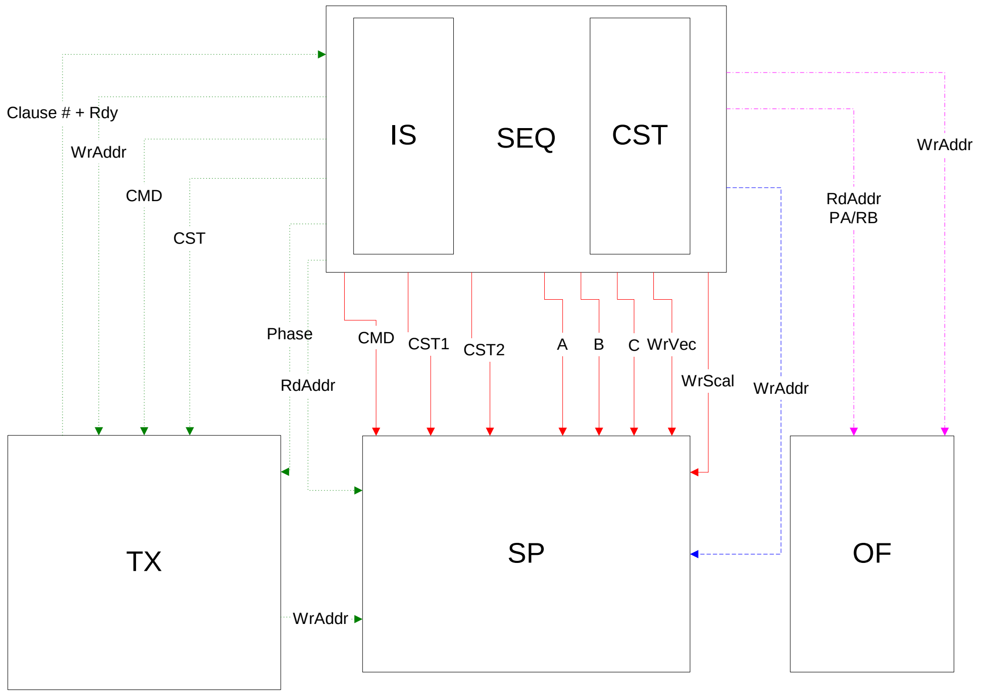
1.2 Data Flow graph


instruction
constants from RE
Register File
MAC
scalar input/output
pipeline stage
texture request
instruction
Register File
MAC
scalar input/output
pipeline stage
texture request
Scalar Unit
instruction
Register File
MAC
scalar input/output
pipeline stage
texture request
texture data
primitive data from RE
instruction
constants from RE
Register File
MAC
texture request
scalar input/output
Mux
texture address
to Primitive Assembly Unit or RenderBackend



The gray area represents blocks that are replicated 4 times per shader pipe (16 times on the overall chip).

1.3 Control Graph



In green is represented the Texture control interface, in red the ALU control interface, in blue the Interpolated/Vector control interface and in purple is the output file control interface.

2. Interpolated data bus

Formatted: Bullets and Numbering

Since each of the register file is actually physically divided (one 32x128 per MAC) and we don't have the place to hold a maximum size vector of vertices in the parameter buffer, we need to interpolate on a parameter basis rather than on a quad basis. So the order to the register file will be:

Q0P0 Q1P0 Q2P0 Q3P0 Q0P1 Q1P1 Q2P1 Q3P2 Q0P3 Q1P3 …

3. Instruction Store

Formatted: Bullets and Numbering

There is going to be only one instruction store for the whole chip. It may contain up to 2000 instructions of 96 bits each. The instruction store is loaded by the sequencer using the memory hub. The read bandwith from this store is 24 bits/clock/pipe. To achieve this this instruction store is likely to be broken up into 4 blocks. An ALU instruction section (1R/1W) split in two and a texture section (1R/1W) also split in two. The bandwith out of those memories is 96 bits/clock.

4. Constant Store

Formatted: Bullets and Numbering

The constant store is managed by the CP. The sequencer is aware of where the constants are using a remaping table also managed by the CP. A likely size for the constant store is 512x128 bits. The constant store is also planned to be shared. The read BW from the constant store is 512/4 bits/clock/pipe and the write bandwith is 32/4 bits/clock.

5. Looping and Branches

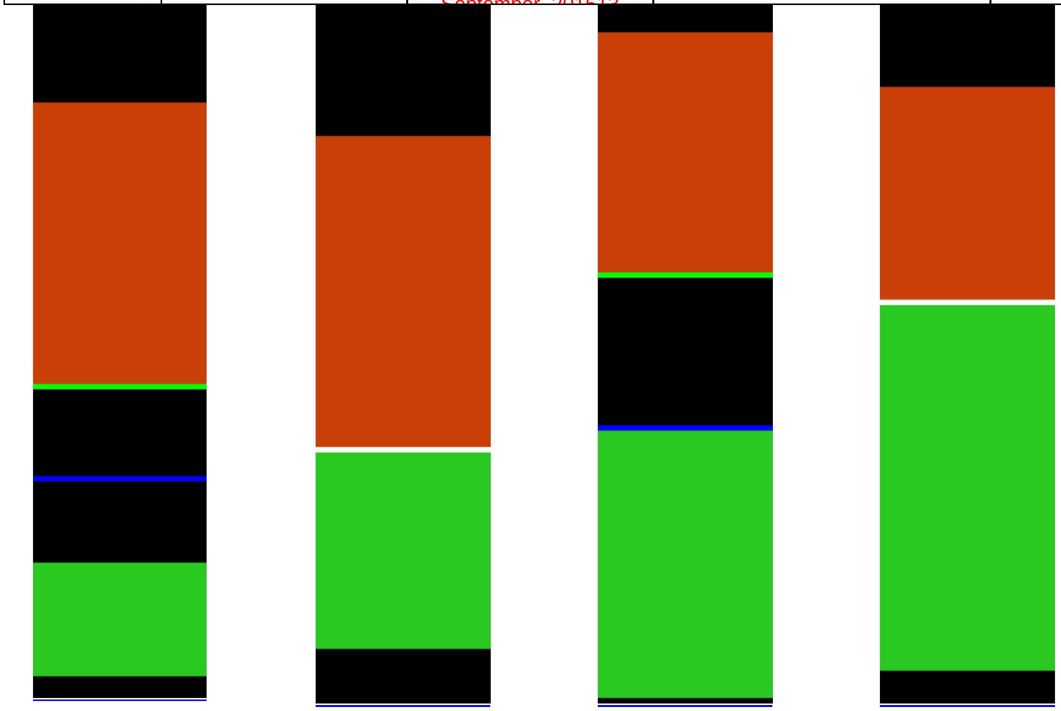
Formatted: Bullets and Numbering

Loops and branches are planned to be supported and will have to be dealt with at the sequencer level. However, it is still unclear if we plan on supporting data dependent branches or not.

6. Register file allocation

Formatted: Bullets and Numbering

The register file allocation for vertices and pixels can either be static or dynamic. In both cases, the register file in managed using two round robins (one for pixels and one for vertices). In the dynamic case the boundary between pixels and vertices is allowed to move, in the static case it is fixed to VERTEX_REG_SIZE for vertices and 256-VERTEX_REG_SIZE for pixels.



Above is an example of how the algorithm works. Vertices come in from top to bottom; pixels come in from bottom to top. Vertices are in orange and pixels in green. The blue line is the tail of the vertices and the green line is the tail of the pixels. Thus anything between the two lines is shared. When pixels meets vertices the line turns white and the boundary is static until both vertices and pixels share the same “unallocated bubble”. Then the boundary as allowed to move again.

Formatted: Bullets and Numbering

~~2.~~7. Texture Arbitration

The texture arbitration logic chooses one of the 8 potentially pending texture clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute. Once chosen, the clause state machine will send one 2x2 texture fetch per clock (or 4 fetches in one clock every 4 clocks) until all the texture fetch instructions of the clause are sent. This means that there cannot be any dependencies between two texture fetches of the same clause.

The arbitrator will not wait for the texture fetches to return prior to selecting another clause for execution. The texture pipe will be able to handle up to ~~100~~X(?) in flight texture fetches and thus there can be a fair number of active clauses waiting for their texture return data.

Formatted: Bullets and Numbering

~~3.~~8. ALU Arbitration

ALU arbitration proceeds in almost the same way than texture arbitration. The ALU arbitration logic chooses one of the 8 potentially pending ALU clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute~~execute. If the packet chosen is a packet of vertices, the state machine issues one instruction every 4 clocks until the clause is finished. This means that the compiler has to insert nops between two dependent successive instructions. If the packet is a pixel packet it is made out of two sub-vectors of 16. Thus the state machine issues the first instruction for the first sub-vector and then, 4 clocks later, the first instruction of the second sub-vector and so on until the clause is finished. ~~. There are two ALU arbitrers, one for the even clocks and

one for the odd clocks. For exemple, here is the sequencing of two interleaved ALU clauses (E and O stands for Even and Odd):

Einst0 Oinst0 Einst1 Oinst1 Einst2 Oinst2 Einst0 Oinst3 Einst1 Oinst4 Einst2 Oinst0…
Proceeding this way hides the latency of 8 clocks of the ALUs.

Formatted: Bullets and Numbering

~~4.~~9. Handling Stalls

When the output file is full, the sequencer prevents the ALU arbitration logic to select the last clause (this way nothing can exit the shader pipe until there is place in the output file. If ~~we have the ability to export at any clause~~the packet is a vertex packet and the position buffer is full (POS_FULL) then the sequencer also prevents a thread to enter the exporting clause (4?). The sequencer will set the OUT_FILE_FULL signal n clocks before the output file is actually full and thus the ALU arbitrer will be able read this signal and act accordingly by not preventing exporting clauses to proceed.

Formatted: Bullets and Numbering

~~5.~~10. Content of the reservation station FIFOs

3 bits of Render State ~~and~~ 6-7 bits for the base address of the instruction store and some bits for LOD correction. Every other information (such as the coverage mask, quad address, etc.) is put in a FIFO and is retrieved when the quad exits the shader pipe to enter in the output file buffer. Since pixels and vertices are kept in order in the shader pipe, we only need two fifos (one for vertices and one for pixels) deep enough to cover the shader pipe latency. This size will be determined later when we will know the size of the small fifos between the reservation stations.

Formatted: Bullets and Numbering

~~6.~~11. The Output File (RB FIFO and Parameter Cache)

The output file is where program results are exported when the pixel/vertex shader finishes. It constists of a 512x128 memory cell that is statically divided between pixels and vertices. ~~Each section is a regular FIFO.~~ The output file has 1 write port and 1 read port. The sequencer is responsible for managing the addresses of this output file and for stalling the shader pipe should this output file fill up. The management is done by keeping the tail and head pointers of each sections (pixels and vertices) and incrementing them using a simple RoundRobin allocation policy. The sequencer must also arbitrate between the PA and the RB for the use of the read port. This arbitration will either be priority based or just interleaved evenly (1 read every 2 clocks for each of the blocks).

Formatted: Bullets and Numbering

~~7.~~12. Interfaces

~~7.1~~12.1 External Interfaces

~~7.1.1~~12.1.1 *Sequencer to Shader Engine Bus*

This is a bus that sends the instruction and constant data to all 4 Sub-Engines of the Shader. Because a new instruction is needed only every 4 clocks, the width of the bus is divided by 4 and both constants and instruction are sent over those 4 clocks.

Name	Direction	Bits	Description
Instruction Start	SEQ-> SP	1	High on first cycle of transfer
Constant 0	SEQ-> SP	32	128 bits transferred over 4 cycles, alpha first…blue last
Constant 1	SEQ-> SP	32	128 bits transferred over 4 cycles, alpha first…blue last
Instruction	SEQ-> SP	30	120 bits transferred over 4 cycles (order TBD) ?

Formatted: Bullets and Numbering

~~7.1.2~~12.1.2 *Shader Engine to Output File*

Every clock each Sub-Engine can output 128 bits of 'vector' data and 32 bits of 'scalar' data to an output file (?). This data will be compressed into 128 bits total prior to storage in output file.

Name	Direction	Bits	Description
UL_Vector_Out	SP-> OF	128	Vector Data out

UL_Scalar_Out	SP-> OF	32	Vector Data out
UR_Vector_Out	SP-> OF	128	Vector Data out
UR_Scalar_Out	SP-> OF	32	Vector Data out

Name	Direction	Bits	Description
LL_Vector_Out	SP-> OF	128	Vector Data out
LL_Scalar_Out	SP-> OF	32	Vector Data out
LR_Vector_Out	SP-> OF	128	Vector Data out
LR_Scalar_Out	SP-> OF	32	Vector Data out

~~7.1.3~~12.1.3 *Shader Engine to Texture Unit Bus (Fast Bus)*

Formatted: Bullets and Numbering

One quad's worth of addresses is transferred to Texture Unit every clock. These are sourced from a different pixel within each of the sub-engines repeating every 4 clocks. The ~~register file~~register file index to read must precede the data by 2 clocks. The Read address associated with Quad 0 must be sent 1 clock after the Instruction Start signal is sent, so that data is read 3 clocks after the Instruction Start.

One Quad's worth of Texture Data may be written to the ~~Register File~~Register file every clock. These are directed to a different pixel of the sub-engines repeating every 4 clocks. The ~~register file~~register file index to write must accompany the data. Data and Index associated with the Quad 0 must be sent 3 clocks after the Instruction Start signal is sent.

Name	Direction	Bits	Description
Tex_Read_Register_Index	SEQ->SP	8	Index into ~~Register File~~Register files for reading Texture Address
Tex_RegFile_Read_Data	SP->TEX	512	4 Texture Addresses read from the ~~Register File~~Register file
Tex_Write_Register_Index	SEQ->TEX	8	Index into ~~Register file~~Register file for write of returned Texture Data

~~7.1.4~~12.1.4 *Sequencer to Texture Unit bus (Slow Bus)*

Formatted: Bullets and Numbering

Once every four clock, the texture unit sends to the sequencer on wich clause it is now working and if the data in the registers is ready or not. This way the sequencer can update the texture counters for the reservation station fifos. The sequencer also provides the intruction and constants for the texture fetch to execute and the address in the ~~register file~~register file where to write the texture return data.

Name	Direction	Bits	Description
Tex_Ready	TEX→ SEQ	1	Data ready
Tex_Clause_Num	TEX→ SEQ	3	Clause number
Tex_cst	SEQ→TEX	?	Texture constants X bits sent over 4 clocks
Tex_Inst	SEQ→TEX	?	Texture fetch instruction X bits sent over 4 clocks

~~7.1.5~~12.1.5 *Shader Engine to RE/PA Bus*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
Interpolator_Register_Index	SEQ->SP	8	Index into ~~Register File~~Register files for write of Interpolator/Index Data
Interpolator_Write_Mask	SEQ->SP	1	Write Mask. The same write mask is used for all 4 pixels
Interpolator_Write_Data	RE/PA->SP	512	4 interpolated vectors or vectors of indices

12.1.6 *PA to sequencer*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
Adress	PA→SEQ	?	Dealocation adress sent by the PA telling the Sequencer that it is now possible to free this space in the parameter buffer. This token is a pointer in the parameter cache and 4 bits to tell the size wich is to be freed up.

Formatted

Formatted

Formatted

Formatted

Formatted: Bullets and Numbering

~~8.~~13. Open issues

There is currently an issue with constants. If the constants are not the same for the whole vector of vertices, we don't have the bandwith from the texture store to feed the ALUs. Two solutions exists for this problem:

1) Let the compiler handle the case and put those instructions in a texture clause so we can use the bandwith there to operate. This requires a significant amount of temporary storage in the register store.
2) Waterfall down the pipe allowing only at a given time the vertices having the same constants to operate in parrallel. This might in the worst case slow us down by a factor of 16.

Need to do some testing on the size of the ~~register file~~register file as well as on the ~~register file~~register file allocation method (dynamic VS static).

~~Ability to export at any clause?~~

Saving power?

~~Are we working on 32 vertices at a time or 16?~~

Size of the fifo containing the information of a vector of pixels/vertices. And size of the fifos before the reservation stations.

Sequencer Instruction memory, and constant memory.

Arbitration policy for the output file.

Loops and branches.

The parameter cache may end up in the PA rather than in the RS. Parameter cache management thus may change.

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	14 August, 2001~~14~~ ~~August, 20017 May~~	4 September, 2015~~24~~ ~~August, 20016 August~~	GEN-CXXXXX-REVA	1 of 20

Author: Laurent Lefebvre

Issue To:	**Copy No:**

R400 Sequencer Specification

SEQ

Version 0.4~~2~~

Overview: This is an architectural specification for the R400 Sequencer block (SEQ). It provides an overview of the required capabilities and expected uses of the block. It also describes the block interfaces, internal sub-blocks, and provides internal state diagrams.

AUTOMATICALLY UPDATED FIELDS:
Document Location: C:\perforce\r400\arch\doc\gfx\RE\R400_Sequencer.doc
Current Intranet Search Title: R400 Sequencer Specification

APPROVALS	
Name/Dept	Signature/Date

Remarks:

THIS DOCUMENT CONTAINS CONFIDENTIAL INFORMATION THAT COULD BE SUBSTANTIALLY DETRIMENTAL TO THE INTEREST OF ATI TECHNOLOGIES INC. THROUGH UNAUTHORIZED USE OR DISCLOSURE.

Table Of Contents

1. OVERVIEW .... 4~~3~~
1.1 Top Level Block Diagram .... 5~~4~~
1.2 Data Flow graph .... 9~~7~~
1.3 Control Graph .... 12~~10~~
2. INTERPOLATED DATA BUS .... 12~~10~~
3. INSTRUCTION STORE .... 12~~10~~
4. CONSTANT STORE .... 13~~11~~
5. LOOPING AND BRANCHES .... 13~~11~~
6. REGISTER FILE ALLOCATION .... 13~~11~~
7. TEXTURE ARBITRATION .... 14~~12~~
8. ALU ARBITRATION .... 14~~12~~
9. HANDLING STALLS .... 15~~13~~
10. CONTENT OF THE RESERVATION STATION FIFOS .... 15~~13~~
11. THE OUTPUT FILE (RB FIFO AND PARAMETER CACHE) .... 15~~13~~
12. INTERFACES .... 15~~13~~
12.1 External Interfaces .... 15~~13~~
12.1.1 Sequencer to Shader Engine Bus .... 15~~13~~
12.1.2 Shader Engine to Output File 15~~13~~
12.1.3 Shader Engine to Texture Unit Bus (Fast Bus) .... 16~~14~~
12.1.4 Sequencer to Texture Unit bus (Slow Bus) 16~~14~~
12.1.5 Shader Engine to RE/PA Bus 16~~14~~
12.1.6 PA? to sequencer .... 16~~14~~
13. EXAMPLES OF PROGRAM EXECUTIONS .... 17~~14~~
13.1.1 Sequencer Control of a Vector of Vertices 17~~14~~
13.1.2 Sequencer Control of a Vector of Pixels 18~~16~~
13.1.3 Notes .... 19~~17~~
14. OPEN ISSUES .... 19~~17~~
~~1. OVERVIEW .... 3~~
~~1.1 Top Level Block Diagram .... 4~~
~~1.2 Data Flow graph .... 7~~
~~1.3 Control Graph .... 10~~
~~2. INTERPOLATED DATA BUS .... 10~~
~~3. INSTRUCTION STORE .... 10~~
~~4. CONSTANT STORE .... 11~~
~~5. LOOPING AND BRANCHES .... 11~~
~~6. REGISTER FILE ALLOCATION .... 11~~
~~7. TEXTURE ARBITRATION .... 12~~
~~8. ALU ARBITRATION .... 12~~
~~9. HANDLING STALLS .... 13~~
~~10. CONTENT OF THE RESERVATION STATION FIFOS .... 13~~
~~11. THE OUTPUT FILE (RB FIFO AND PARAMETER CACHE) .... 13~~
~~12. INTERFACES .... 13~~
~~12.1 External Interfaces .... 13~~
~~12.1.1 Sequencer to Shader Engine Bus .... 13~~
~~12.1.2 Shader Engine to Output File 13~~
~~12.1.3 Shader Engine to Texture Unit Bus (Fast Bus) .... 14~~
~~12.1.4 Sequencer to Texture Unit bus (Slow Bus) 14~~
~~12.1.5 Shader Engine to RE/PA Bus 14~~
~~12.1.6 PA to sequencer .... 14~~
~~13. OPEN ISSUES .... 14~~
~~1. OVERVIEW .... 3~~
~~1.1 Top Level Block Diagram .... 4~~
~~2. TEXTURE ARBITRATION .... 9~~
~~3. ALU ARBITRATION .... 9~~
~~4. HANDLING STALLS .... 10~~
~~5. CONTENT OF THE RESERVATION STATION FIFOS .... 10~~
~~6. INTERFACES .... 10~~
~~6.1 External Interfaces .... 10~~
~~6.1.1 Sequencer to Shader Engine Bus 10~~
~~6.1.2 Shader Engine to Output File 10~~
~~6.1.3 Shader Engine to Texture Unit Bus (Fast Bus) .... 10~~

~~6.1.4 Sequencer to Texture Unit bus (Slow Bus)... 11~~
~~6.1.5 Shader Engine to RE/PA Bus .. 11~~
~~7. OPEN ISSUES................................ 11~~

Revision Changes:

Rev 0.1 (Laurent Lefebvre) Date: May 7, 2001	First draft.
Rev 0.2 (Laurent Lefebvre) Date : July 9, 2001	Changed the interfaces to reflect the changes in the SP. Added some details in the arbitration section.
Rev 0.3 (Laurent Lefebvre) Date : August 6, 2001	Reviewed the Sequencer spec after the meeting on August 3, 2001.
Rev 0.4 (Laurent Lefebvre) Date : August 24, 2001	Added the dynamic allocation method for register file and an example (written in part by Vic) of the flow of pixels/vertices in the sequencer.

1. Overview

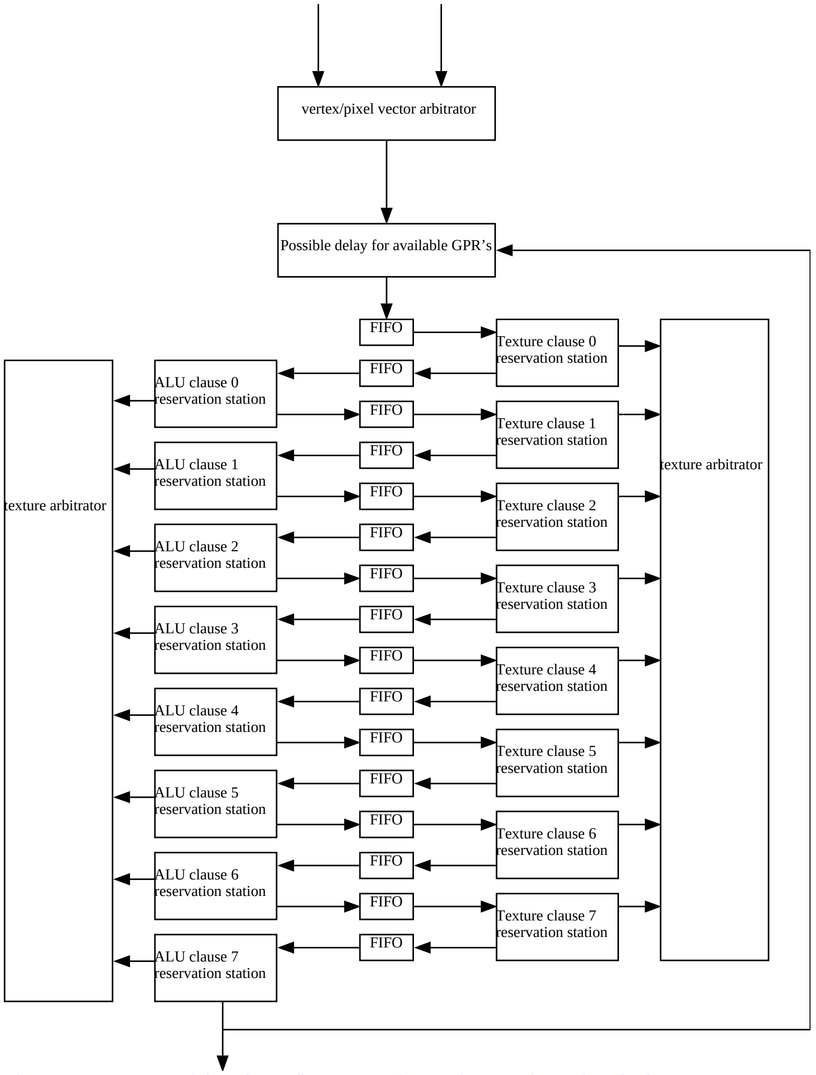
The sequencer first arbitrates between vectors of 16~~ (maybe 32)~~ vertices that arrive directly from primitive assembly and vectors of ~~8~~4 quads (16 pixels) ~~(32 pixels)~~ that are generated in the raster engine.

The vertex or pixel program specifies how many GPR's it needs to execute. The sequencer will not start the next vector until the needed space is available.

The sequencer is based on the R300 design. It chooses ~~an~~ two ALU clauses and a texture clause to execute, and executes all of the instructions in ~~a~~a clause before looking for a new clause of the same type. Two ALU clauses are executed interleaved to hide the ALU latency. Each vector will have eight texture and eight ALU clauses, but clauses do not need to contain instructions. A vector of pixels or vertices ping-pongs along the sequencer FIFO, bouncing from texture reservation station to alu reservation station. A FIFO exists between each reservation stage, holding up vectors until the vector currently occupying a reservation station has left. A vector at a reservation station can be chosen to execute. The sequencer looks at all eight alu reservation stations to choose an alu clause to execute and all eight texture stations to choose a texture clause to execute. The arbitrator will give priority to clauses/reservation stations closer to the ~~top~~ bottom of the pipeline. It will not execute an alu clause until the texture fetches initiated by the previous texture clause have completed. There are two separate sets of reservation stations, one for pixel vectors and one for vertices vectors. This way a pixel can pass a vertex and a vertex can pass a pixel.

To support the shader pipe the raster engine also contains the shader instruction cache and constant store. There are only one constant store for the whole chip and one instruction store. These will be shared among the four shader pipes.

1.1 Top Level Block Diagram



vertex/pixel vector arbitrator
Possible delay for available GPR's
FIFO
Texture clause 0 reservation station
Texture clause 1 reservation station
Texture clause 2 reservation station
Texture clause 3 reservation station
Texture clause 4 reservation station
Texture clause 5 reservation station
Texture clause 6 reservation station
Texture clause 7 reservation station
ALU clause 0 reservation station
ALU clause 1 reservation station
ALU clause 2 reservation station
ALU clause 3 reservation station
ALU clause 4 reservation station
ALU clause 5 reservation station
ALU clause 6 reservation station
ALU clause 7 reservation station
texture arbitrator
texture arbitrator

There are two sets of the above figure, one for vertices and one for pixels.

The rasterizer always checks the vertices FIFO first and if allowed by the sequencer sends the data to the shader. If the vertex FIFO is empty then, the rasterizer takes the first entry of the pixel FIFO (a vector of ~~32~~ 16 pixels) and sends it to the interpolators. Then the sequencer takes control of the packet. The packet consists of 3 bits of state, 6-7 bits for the base address of the Shader program and some information on the coverage to determine texture LOD. All other information (2x2 adresses) is put in a FIFO (one for the pixels and one for the vertices) and retrieved when the packet finishes its last clause.

On receipt of a packet, the input state machine (not pictured but just before the first FIFO) allocated enough space in the registers to store the interpolated values and temporaries. Following this, the input state machine stacks the packet in the first FIFO.

On receipt of a command, the level 0 texture machine issues a texure request and corresponding register address for the texture address (ta). A small command (tcmd) is passed to the texture system identifying the current level number (0) as well as the register ~~set being used~~write address for the texture return data. One texture request is sent every 4 clocks causing the texturing of four 2x2s worth of data (or 16 vertices). Once all the requests are sent the packet is put in FIFO 1.

Upon recept of the return data~~ (identified by the tcmd containing the level number 0), the level 0 texture machine issues a register address for the return value (td)~~, the texture unit writes the data to the register file using the write address that was provided by the level 0 texture machine and sends the clause number (0) to the level 0 texture state machine to signify that the write is done and thus the data is ready. Then, the level 0 texture machine~~ it~~ increments the counter of FIFO ~~one~~ 1 to signify to the ALU 1 that the data is ready to be processed.

On receipt of a command, the level 0 ALU machine first decrements the input FIFO counter and then issues a complete set of level 0 shader instructions. For each instruction, the state machine generates 3 source addresses, one destination address (~~2~~ 3 cycles later) and an instruction~~ id wich is used to index into the instruction store~~. Once the last instruction as been issued, the packet is put into FIFO 2.~~ Note that in the case of a pixel packet, the two vectors of 16 pixels are consecutive in order to hide the latency of the ALUs (8 cycles).~~

There will always be two active ALU clauses at any given time (and two arbitrers)~~ In this case, the instructions of a vector are interleaved with the instructions of the other vector.~~ One arbiter will arbitrate over the odd clock cycles and the other one will arbitrate over the even clock cycles. The only constraints between the two arbitrers is that they are not allowed to pick the same clause number as they other one is currently working on if the packet os of the same type.

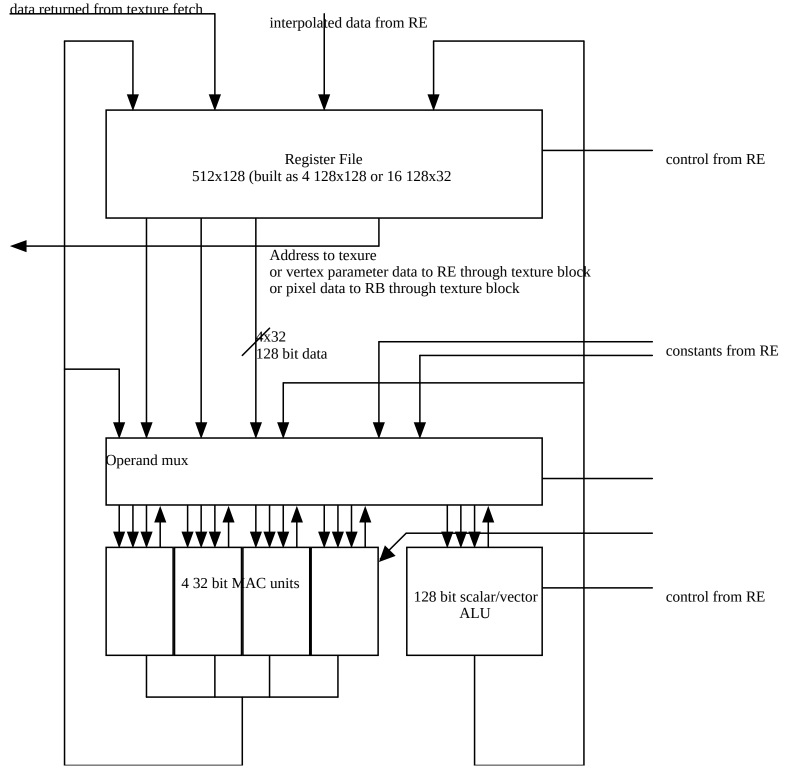
Formatted

If the packet is a vertex packet, upon reaching ALU clause 4, it can export the position if the position is ready. So the arbitrer must prevent ALU clause 4 to be selected if the positional buffer is full (or can't be accessed). Along with the positional data, the location where the vertex data is to be put is also sent (parameter data pointers).

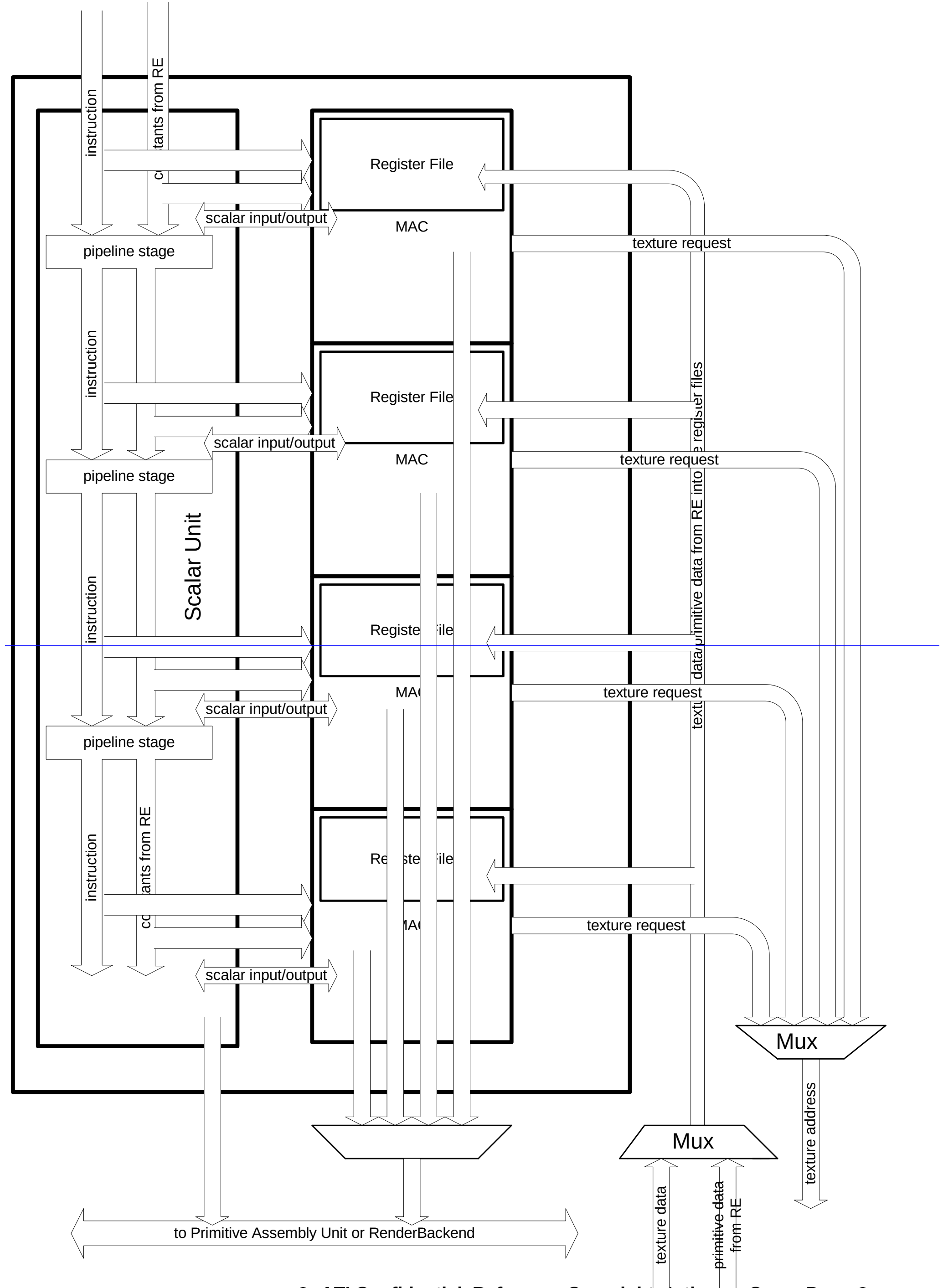
All other level process in the same way until the packet finally reaches the last ALU machine (8). On completion of the level 8 ALU clause, a valid bit is sent to the Render Backend which picks up the color data. This requires that the last instruction writes to the output register – a condition that is almost always true. If the packet was a vertex packet, instead of sending the valid bit to the RB, it is sent to the PA~~, which picks up the data and puts it into the vertex store~~ so it can know that the data present in the parameter store is valid.

Only ~~one~~ two ALU state machine may have access to the ~~SRAM~~register file address bus or the instruction decode bus at one time. Similarly, only one texture state machine may have access to the ~~SRAM~~register file address bus at one time. Arbitration is performed by ~~two~~ three arbitrer blocks (~~one~~ two for the ALU state machines and one for the texture state machines). The arbitrers always favor the higher number state machines, preventing a bunch of half finished jobs from clogging up the ~~SRAM~~register ~~S~~files.

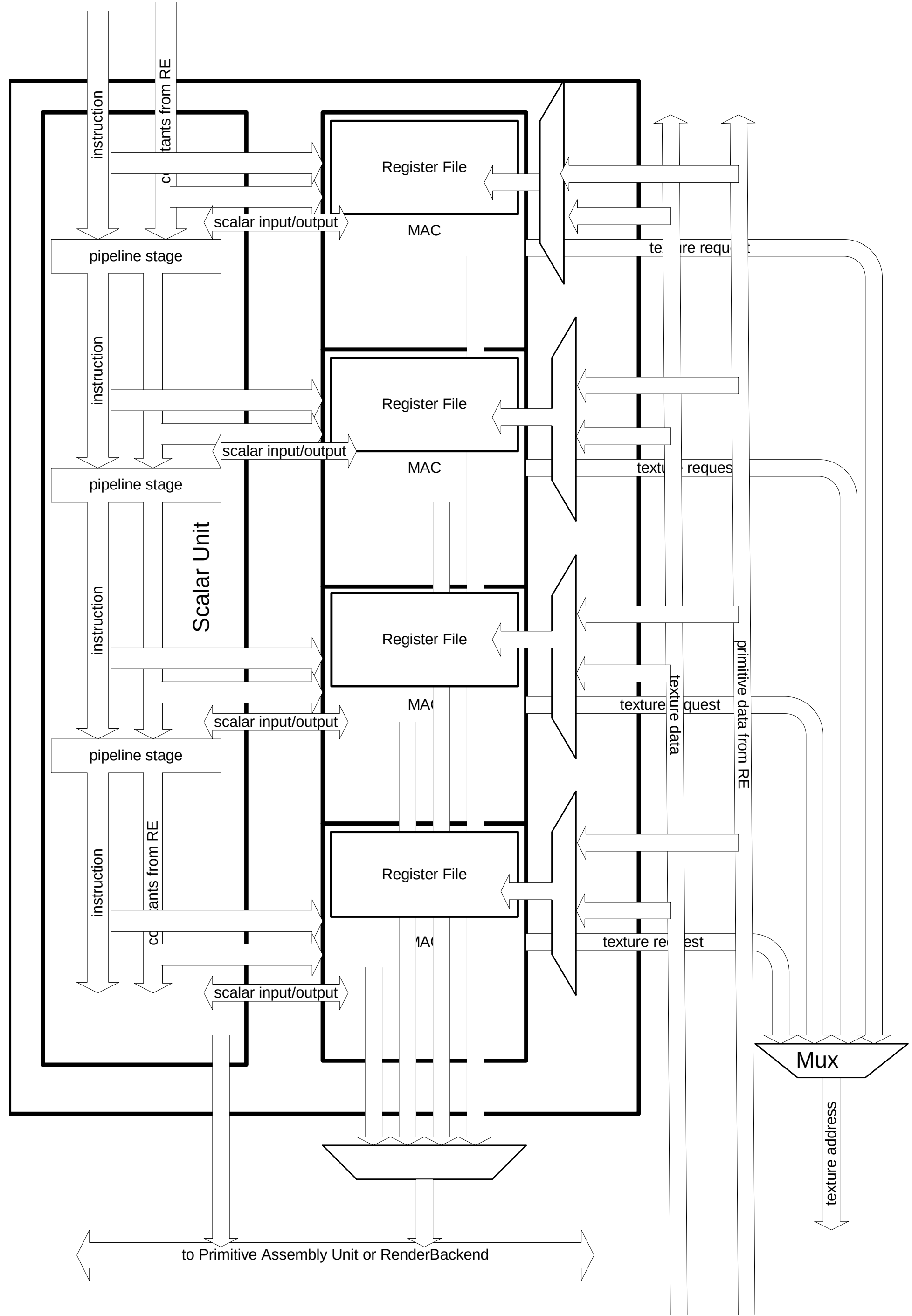
Each state machine maintains an address pointer specifying where the 16~~ (or 32)~~ entries vector is located in the ~~SRAM~~register file (the texture machine has two pointers one for the read address and one for the write). Upon completion of its job, the address pointer is incremented by a predefined amount equal to the total number of registers required by the shading code. A comparison of the address pointer for the first state machine in the chain (the input state machine), and the last machine in the chain (the level 8 ALU machine), gives an indication of how much unallocated ~~SRAM~~register file memory is available



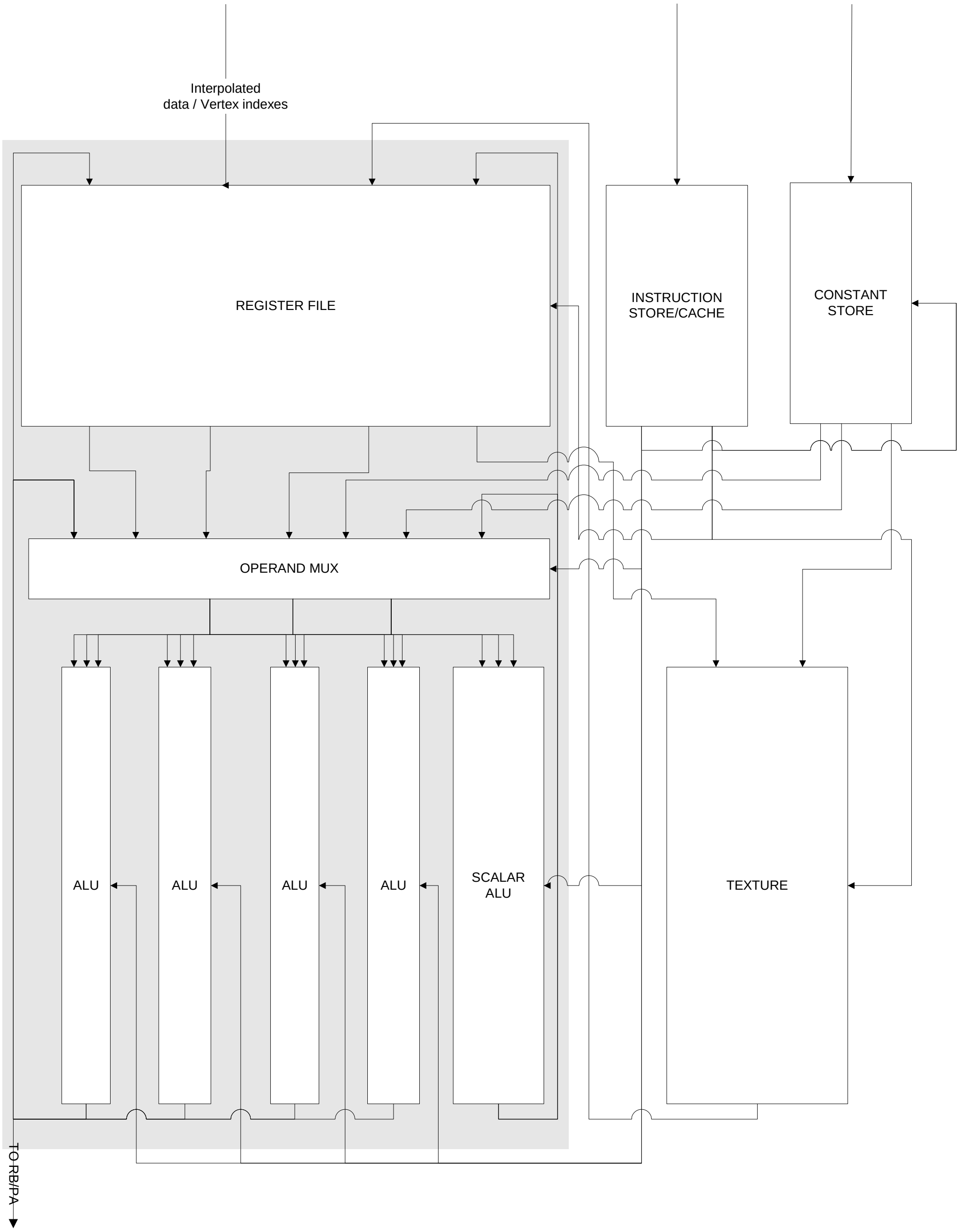
data returned from texture fetch
interpolated data from RE
Register File
512x128 (built as 4 128x128 or 16 128x32
control from RE
Address to texure
or vertex parameter data to RE through texture block
or pixel data to RB through texture block
4x32
128 bit data
constants from RE
Operand mux
4 32 bit MAC units
128 bit scalar/vector
ALU
control from RE


instruction
constants from RE
Register File
scalar input/output
MAC
pipeline stage
texture request
instruction
Register File
scalar input/output
MAC
pipeline stage
texture request
Scalar Unit
instruction
Register File
MAC
scalar input/output
pipeline stage
texture request
texture data/primitive data from RE into the register files
instruction
constants from RE
Register File
MAC
texture request
scalar input/output
Mux
texture address
Mux
to Primitive Assembly Unit or RenderBackend
texture data
primitive data from RE

1.2 Data Flow graph

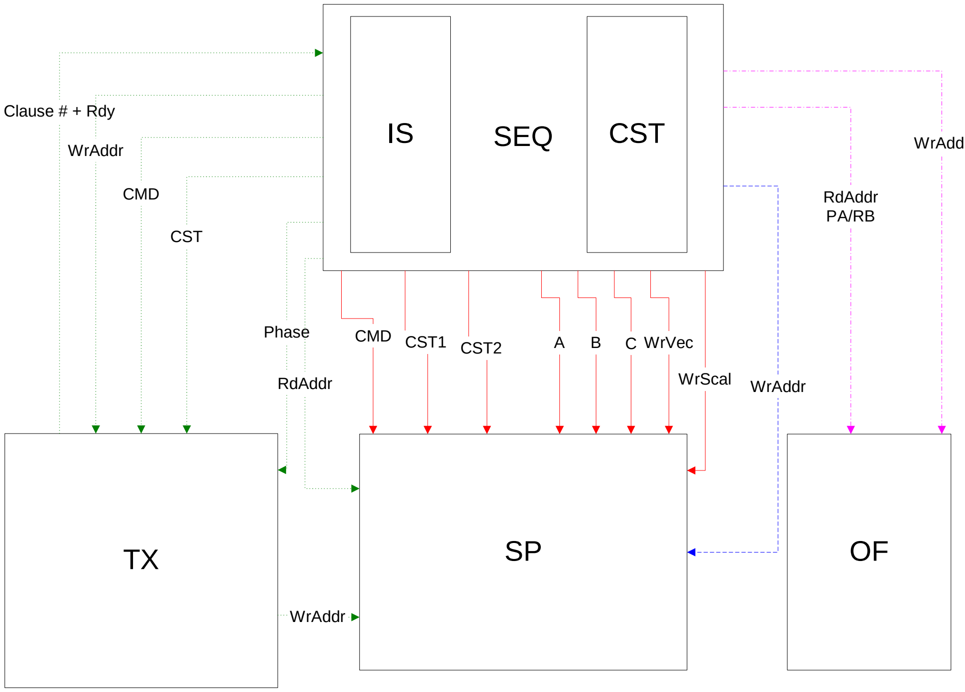


instruction
constants from RE
Register File
scalar input/output
MAC
pipeline stage
texture request
instruction
Register File
scalar input/output
MAC
pipeline stage
texture request
Scalar Unit
instruction
Register File
MAC
scalar input/output
texture request
texture data
primitive data from RE
pipeline stage
instruction
constants from RE
Register File
MAC
texture request
scalar input/output
Mux
texture address
to Primitive Assembly Unit or RenderBackend



The gray area represents blocks that are replicated 4 times per shader pipe (16 times on the overall chip).

1.3 Control Graph



In green is represented the Texture control interface, in red the ALU control interface, in blue the Interpolated/Vector control interface and in purple is the output file control interface.

Formatted: Bullets and Numbering

2. Interpolated data bus

Since each of the register file is actually physically divided (one 128x128 per MAC) and we don't have the place to hold a maximum size vector of vertices in the parameter buffer, we need to interpolate on a parameter basis rather than on a quad basis. So the order to the register file will be:

Q0P0 Q1P0 Q2P0 Q3P0 Q0P1 Q1P1 Q2P1 Q3P1 Q0P2 Q1P2 …

Formatted: Bullets and Numbering

3. Instruction Store

There is going to be only one instruction store for the whole chip. It may contain up to 2000 instructions of 96 bits each.

{ISSUE : The instruction store is loaded by the sequencer using the memory hub ?}.

The read bandwith from this store is 24 bits/clock/pipe. To achieve this this instruction store is likely to be broken up into 4 blocks. An ALU instruction section (1R/1W) split in two and a texture section (1R/1W) also split in two. The bandwith out of those memories is 96 bits/clock.

4. Constant Store

Formatted: Bullets and Numbering

The constant store is managed by the CP. The sequencer is aware of where the constants are using a remaping table also managed by the CP. A likely size for the constant store is 512x128 bits. The constant store is also planned to be shared. The read BW from the constant store is 512/4 bits/clock/pipe and the write bandwith is 32/4 bits/clock.

5. Looping and Branches

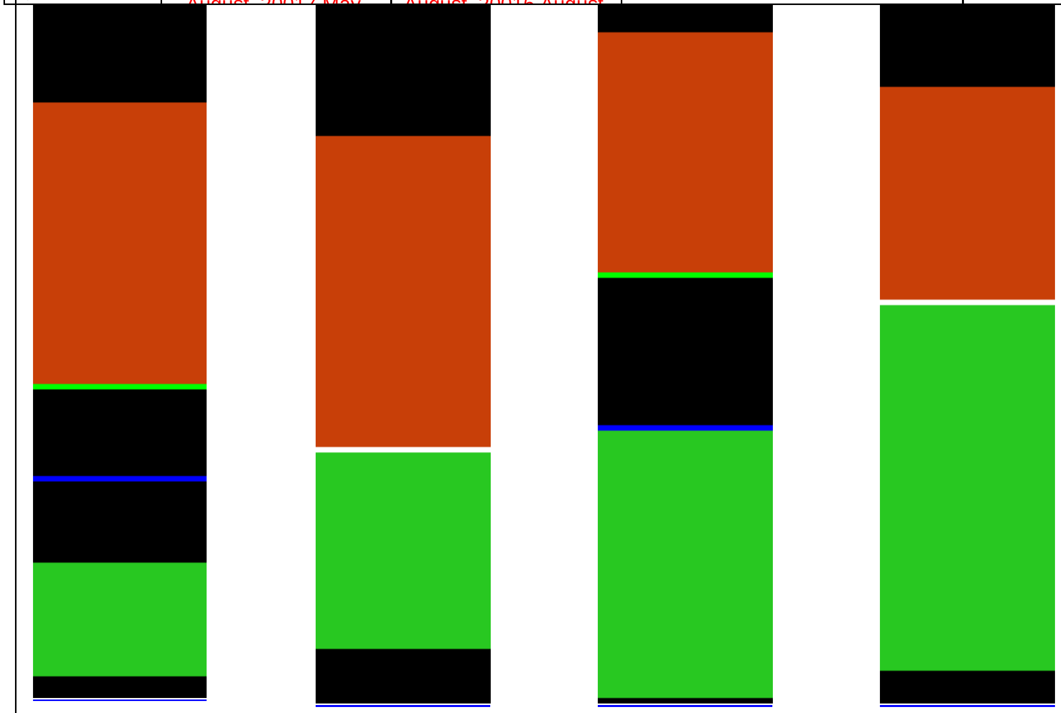
Formatted: Bullets and Numbering

Loops and branches are planned to be supported and will have to be dealt with at the sequencer level. However, it is still unclear if we plan on supporting data dependent branches or not.

6. Register file allocation

Formatted: Bullets and Numbering

The register file allocation for vertices and pixels can either be static or dynamic. In both cases, the register file in managed using two round robins (one for pixels and one for vertices). In the dynamic case the boundary between pixels and vertices is allowed to move, in the static case it is fixed to VERTEX_REG_SIZE for vertices and 256-VERTEX_REG_SIZE for pixels.



Above is an example of how the algorithm works. Vertices come in from top to bottom; pixels come in from bottom to top. Vertices are in orange and pixels in green. The blue line is the tail of the vertices and the green line is the tail of the pixels. Thus anything between the two lines is shared. When pixels meets vertices the line turns white and the boundary is static until both vertices and pixels share the same “unallocated bubble”. Then the boundary as allowed moving again.

Formatted: Bullets and Numbering

~~2.~~7. Texture Arbitration

The texture arbitration logic chooses one of the 8 potentially pending texture clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute. Once chosen, the clause state machine will send one 2x2 texture fetch per clock (or 4 fetches in one clock every 4 clocks) until all the texture fetch instructions of the clause are sent. This means that there cannot be any dependencies between two texture fetches of the same clause.

The arbitrator will not wait for the texture fetches to return prior to selecting another clause for execution. The texture pipe will be able to handle up to ~~100~~X(?) in flight texture fetches and thus there can be a fair number of active clauses waiting for their texture return data.

Formatted: Bullets and Numbering

~~3.~~8. ALU Arbitration

ALU arbitration proceeds in almost the same way than texture arbitration. The ALU arbitration logic chooses one of the 8 potentially pending ALU clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute~~execute. If the packet chosen is a packet of vertices, the state machine issues one instruction every 4 clocks until the clause is finished. This means that the compiler has to insert nops between two dependent successive instructions. If the packet is a pixel packet it is made out of two sub-vectors of 16. Thus the state machine issues the first instruction for the first sub-vector and then, 4 clocks later, the first instruction of the second sub-vector and so on until the clause is finished. ~~. There are two ALU arbitrers, one for the even clocks and

one for the odd clocks. For exemple, here is the sequencing of two interleaved ALU clauses (E and O stands for Even and Odd):

Einst0 Oinst0 Einst1 Oinst1 Einst2 Oinst2 Einst0 Oinst3 Einst1 Oinst4 Einst2 Oinst0…
Proceeding this way hides the latency of 8 clocks of the ALUs.

Formatted: Bullets and Numbering

~~4.~~9. Handling Stalls

When the output file is full, the sequencer prevents the ALU arbitration logic to select the last clause (this way nothing can exit the shader pipe until there is place in the output file. If ~~we have the ability to export at any clause~~the packet is a vertex packet and the position buffer is full (POS_FULL) then the sequencer also prevents a thread to enter the exporting clause (4?). The sequencer will set the OUT_FILE_FULL signal n clocks before the output file is actually full and thus the ALU arbitrer will be able read this signal and act accordingly by not preventing exporting clauses to proceed.

Formatted: Bullets and Numbering

~~5.~~10. Content of the reservation station FIFOs

3 bits of Render State ~~and~~ 6-7 bits for the base address of the instruction store and some bits for LOD correction. Every other information (such as the coverage mask, quad address, etc.) is put in a FIFO and is retrieved when the quad exits the shader pipe to enter in the output file buffer. Since pixels and vertices are kept in order in the shader pipe, we only need two fifos (one for vertices and one for pixels) deep enough to cover the shader pipe latency. This size will be determined later when we will know the size of the small fifos between the reservation stations.

Formatted: Bullets and Numbering

~~6.~~11. The Output File (RB FIFO and Parameter Cache)

The output file is where program results are exported when the pixel/vertex shader finishes. It constists of a 512x128 memory cell that is statically divided between pixels and vertices. ~~Each section is a regular FIFO.~~ The output file has 1 write port and 1 read port. The sequencer is responsible for managing the addresses of this output file and for stalling the shader pipe should this output file fill up. The management is done by keeping the tail and head pointers of each sections (pixels and vertices) and incrementing them using a simple RoundRobin allocation policy. The sequencer must also arbitrate between the PA and the RB for the use of the read port. This arbitration will either be priority based or just interleaved evenly (1 read every 2 clocks for each of the blocks).

Formatted: Bullets and Numbering

~~7.~~12. Interfaces

~~7.1~~12.1 External Interfaces

~~7.1.1~~12.1.1 *Sequencer to Shader Engine Bus*

This is a bus that sends the instruction and constant data to all 4 Sub-Engines of the Shader. Because a new instruction is needed only every 4 clocks, the width of the bus is divided by 4 and both constants and instruction are sent over those 4 clocks.

Name	**Direction**	**Bits**	**Description**
Instruction Start	SEQ-> SP	1	High on first cycle of transfer
Constant 0	SEQ-> SP	32	128 bits transferred over 4 cycles, alpha first…blue last
Constant 1	SEQ-> SP	32	128 bits transferred over 4 cycles, alpha first…blue last
Instruction	SEQ-> SP	30	120 bits transferred over 4 cycles (order TBD) ?

Formatted: Bullets and Numbering

~~7.1.2~~12.1.2 *Shader Engine to Output File*

Every clock each Sub-Engine can output 128 bits of 'vector' data and 32 bits of 'scalar' data to an output file (?). This data will be compressed into 128 bits total prior to storage in output file.

Name	**Direction**	**Bits**	**Description**
UL_Vector_Out	SP-> OF	128	Vector Data out

UL_Scalar_Out	SP-> OF	32	Vector Data out
UR_Vector_Out	SP-> OF	128	Vector Data out
UR_Scalar_Out	SP-> OF	32	Vector Data out

Name	Direction	Bits	Description
LL_Vector_Out	SP-> OF	128	Vector Data out
LL_Scalar_Out	SP-> OF	32	Vector Data out
LR_Vector_Out	SP-> OF	128	Vector Data out
LR_Scalar_Out	SP-> OF	32	Vector Data out

~~7.1.3~~12.1.3 *Shader Engine to Texture Unit Bus (Fast Bus)*

Formatted: Bullets and Numbering

One quad's worth of addresses is transferred to Texture Unit every clock. These are sourced from a different pixel within each of the sub-engines repeating every 4 clocks. The ~~register file~~register file index to read must precede the data by 2 clocks. The Read address associated with Quad 0 must be sent 1 clock after the Instruction Start signal is sent, so that data is read 3 clocks after the Instruction Start.

One Quad's worth of Texture Data may be written to the ~~Register File~~Register file every clock. These are directed to a different pixel of the sub-engines repeating every 4 clocks. The ~~register file~~register file index to write must accompany the data. Data and Index associated with the Quad 0 must be sent 3 clocks after the Instruction Start signal is sent.

Name	Direction	Bits	Description
Tex_Read_Register_Index	SEQ->SP	8	Index into ~~Register File~~Register files for reading Texture Address
Tex_RegFile_Read_Data	SP->TEX	512	4 Texture Addresses read from the ~~Register File~~Register file
Tex_Write_Register_Index	SEQ->TEX	8	Index into ~~Register file~~Register file for write of returned Texture Data

~~7.1.4~~12.1.4 *Sequencer to Texture Unit bus (Slow Bus)*

Formatted: Bullets and Numbering

Once every four clock, the texture unit sends to the sequencer on wich clause it is now working and if the data in the registers is ready or not. This way the sequencer can update the texture counters for the reservation station fifos. The sequencer also provides the intruction and constants for the texture fetch to execute and the address in the ~~register file~~register file where to write the texture return data.

Name	Direction	Bits	Description
Tex_Ready	TEX→ SEQ	1	Data ready
Tex_Clause_Num	TEX→ SEQ	3	Clause number
Tex_cst	SEQ→TEX	?	Texture constants X bits sent over 4 clocks
Tex_Inst	SEQ→TEX	?	Texture fetch instruction X bits sent over 4 clocks

~~7.1.5~~12.1.5 *Shader Engine to RE/PA Bus*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
Interpolator_Register_Index	SEQ->SP	8	Index into ~~Register File~~Register files for write of Interpolator/Index Data
Interpolator_Write_Mask	SEQ->SP	1	Write Mask. The same write mask is used for all 4 pixels
Interpolator_Write_Data	RE/PA->SP	512	4 interpolated vectors or vectors of indices

12.1.6 *PA? to sequencer*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
Adress	PA→SEQ	?	Dealocation adress sent by the PA telling the Sequencer that it is now possible to free this space in the parameter buffer. This token is a pointer in the parameter cache and 4 bits to tell the size wich is to be freed up.

Formatted

Formatted

Formatted

Formatted

Formatted: Bullets and Numbering

13. Examples of program executions

13.1.1 *Sequencer Control of a Vector of Vertices*

1. PA sends a vector of 16 vertices (actually vertex indices – 32 bits/index for 512 bit total) to the RE's Vertex FIFO
   - state pointer as well as tag into position cache is sent along with vertices
   - space was allocated in the position cache for transformed position before the vector was sent
   - **also before the vector is sent to the RE, the CP has loaded the global instruction store with the vertex shader program (using the MH?)**
   - The vertex program is assumed to be loaded when we receive the vertex vector.
     - the SEQ then accesses the IS base for this shader using the local state pointer (provided to all sequencers by the RBBM when the CP is done loading the program)
2. SEQ arbitrates between the Pixel FIFO and the Vertex FIFO – basically the Vertex FIFO always has priority
   - at this point the vector is removed from the Vertex FIFO
   - the arbitrer is not going to select a vector to be transformed if the parameter cache is full unless the pipe as nothing else to do (ie no pixels are in the pixel fifo).
3. SEQ allocates space in the SP register file for index data plus GPRs used by the program
   - the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer that came down with the vertices
   - SEQ will not send vertex data until space in the register file has been allocated
4. SEQ sends the vector to the SP register file over the RE_SP interface (which has a bandwidth of 512 bits/cycle)
   - the 16 vertex indices are sent to the 16 register files over 4 cycles
     - RF0 of SU0, SU1, SU2, and SU3 is written the first cycle
     - RF1 of SU0, SU1, SU2, and SU3 is written the second cycle
     - RF2 of SU0, SU1, SU2, and SU3 is written the third cycle
     - RF3 of SU0, SU1, SU2, and SU3 is written the fourth cycle
   - the index is written to the least significant 32 bits **(floating point format?) (what about compound indices)** of the 128-bit location within the register file (w); the remaining data bits are set to zero (x, y, z)
5. SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of texture state machine 0, or TSM0 FIFO)
   - the control packet contains the state pointer, the tag to the position cache and a register file base pointer.
6. TSM0 accepts the control packet and fetches the instructions for texture clause 0 from the global instruction store
   - TSM0 was first selected by the TSM arbiter before it could start
7. all instructions of texture clause 0 are issued by TSM0
8. the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)
   - TSM0 does not wait for requests made to the Texture Unit to complete; it passes the register file write index for the texture data to the TU, which will write the data to the RF as it is received
   - once the TU has written all the data to the register files, it increments a counter that is associated with ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go ahead start to execute the ALU clause
9. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store
10. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of texture state machine 1, or TSM1 FIFO)
11. the control packet continues to travel down the path of reservation stations until all clauses have been executed
    - position can be exported in ALU clause 3 (or 4?); the data (and the tag) is sent over a position bus (which is shared with all four shader pipes) back to the PA's position cache
    - A parameter cache pointer is also sent along with the position data. This tells to the PA where the data is going to be in the parameter cache.
      - there is a position export FIFO in the SP that buffers position data before it gets sent back to the PA

    - the ASM arbiter will prevent a packet from starting an exporting clause if the position export FIFO is full
- parameter data is exported in clause 7 (as well as position data if it was not exported earlier)
    - parameter data is sent to the Parameter Cache over a dedicated bus
    - the SEQ allocates storage in the Parameter Cache, and the SEQ deallocates that space when there is no longer a need for the parameters (it is told by the PA when using a token).
    - the ASM arbiter will prevent a packet from starting on ASM7 if the parameter cache (or the position buffer if position is being exported) is full

12. after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program

13.1.2 *Sequencer Control of a Vector of Pixels*

1. **As with vertex shader programs, pixel shaders are loaded into the global instruction store by the CP**
    - At this point it is assumed that the pixel program is loaded into the instruction store and thus ready to be read.
2. the RE's Parameter Buffer is loaded from the Parameter Cache before the SEQ takes control of the vector
    - after the HZ culling stage a request is made by the RE to send parameter data to the Parameter buffer
    - the Parameter buffer is wide enough to source 3 vertices worth of a particular parameter in one cycle
    - **at this moment the right sequencer will free up the parameter store locations not used anymore using the token provided by the PA**.
3. the RE's Pixel FIFO is loaded with the barycentric coordinates for pixel quads by the detailed walker
    - the state pointer and the LOD correction bits are also placed in the Pixel FIF0
    - the Pixel FIFO is wide enough to source one quad's worth of barycentrics per cycle
4. SEQ arbitrates between Pixel FIFO and Vertex FIFO – when there are no vertices pending OR there is no space left in the register files for vertices, the Pixel FIFO is selected
5. SEQ allocates space in the SP register file for all the GPRs used by the program
    - the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer
    - SEQ will not allow interpolated data to be sent to the shader until space in the register file has been allocated
6. SEQ control starts with the interpolation of parameters (up to 16 per thread) by sending the barycentric coordinates from the Pixel FIFO and the parameters from the Parameter Buffer to the interpolator
    - P0i, P0j, and P0k (the value of P0 at each vertex) are loaded into the interpolator from the Parameter buffer
    - Q0 i, j, and k are loaded into the interpolator from the Pixel FIFO
    - The interpolator then generates the parameter value for each pixel in Q0 (Q0P0)
    - **P0i, P0j, and P0k are sent to the interpolator for Q1 only if Q1 is from a different primitive; if Q1 is from the same primitive as Q0, then the P0i, P0j, and P0k values loaded for Q0 are held by the interpolator and reused for Q1**
        - **a "different_prim" control bit is passed with the barycentric data for each quad in the Pixel FIFO that indicates whether new parameter data needs to be loaded into the interpolator**
    - Q1 i, j, and k are then loaded into the interpolator from the Pixel FIFO
    - The interpolator then generates the parameter value for each pixel in Q1 (Q1P0)
    - Q2P0 and Q3P0 are generated in a similar manner
    - The next set of parameter data - P1i, P1j, and P1k - is then loaded into the interpolator
    - Q0 i, j, and k now must be re-read from the Pixel FIFO – this means that the output of the Pixel FIFO loops through the top four entries on each read command until at the end a final "block_pop" signal is asserted, causing the top four sets of barycentric coordinates to finally be removed
    - so the order of parameter info generated is Q0P0, Q1P0, Q2P0, Q3P0, Q0P1, Q1P1, etc.
7. SEQ controls the transfer of interpolated data to the SP register file over the RE_SP interface (which has a bandwidth of 512 bits/cycle)
    - 16 pixels worth of interpolated parameter data is sent to the 16 register files over 4 cycles
        - RF0 of SU0, SU1, SU2, and SU3 is written with Q0P0 the first cycle
        - RF1 of SU0, SU1, SU2, and SU3 is written with Q1P0 second cycle
        - RF2 of SU0, SU1, SU2, and SU3 is written with Q2P0 third cycle
        - RF3 of SU0, SU1, SU2, and SU3 is written with Q3P0 fourth cycle

8. SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of texture state machine 0, or TSM0 FIFO)
   - note that there is a separate set of reservation stations/arbiters/state machines for vertices and for pixels
   - the control packet contains the state pointer, the register file base pointer, and the LOD correction bits
   - all other informations (such as quad address for example) travels in a separate FIFO
9. TSM0 accepts the control packet and fetches the instructions for texture clause 0 from the global instruction store
   - TSM0 was first selected by the TSM arbiter before it could start
10. all instructions of texture clause 0 are issued by TSM0
11. the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)
    - TSM0 does not wait for texture requests made to the Texture Unit to complete; it passes the register file write index for the texture data to the TU, which will write the data to the RF as it is received
    - once the TU has written all the data for a particular clause to the register files, it increments a counter that is associated with the ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go ahead and pop the FIFO and start to execute the ALU clause
12. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store
13. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of texture state machine 1, or TSM1 FIFO)
14. the control packet continues to travel down the path of reservation stations until all clauses have been executed
    - pixel data is exported in the last ALU clause (clause 7)
      - it is sent to an output FIFO where it will be picked up by the render backend
      - the ASM arbiter will prevent a packet from starting on ASM7 if the output FIFO is full
15. after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program

13.1.3 *Notes*

Formatted: Bullets and Numbering

16. the state machines and arbitrers will operate ahead of time so that they will be able to immediately start the real threads or stall.
17. the register file base pointer for a vector needs to travel with the vector through the reservation stations, but the instruction store base pointer does not – this is because the RF pointer is different for all threads, but the IS pointer is only different for each state and thus can be accessed via the state pointer
18. Waterfalling, parameter buffer allocation, loops and branches and parameter cache de-allocation still needs to be specked out.

~~8.~~14. Open issues

There is currently an issue with constants. If the constants are not the same for the whole vector of vertices, we don't have the bandwith from the texture store to feed the ALUs. Two solutions exists for this problem:

1) Let the compiler handle the case and put those instructions in a texture clause so we can use the bandwith there to operate. This requires a significant amount of temporary storage in the register store.
2) Waterfall down the pipe allowing only at a given time the vertices having the same constants to operate in parrallel. This might in the worst case slow us down by a factor of 16.

Need to do some testing on the size of the ~~register file~~register file as well as on the ~~register file~~register file allocation method (dynamic VS static).

~~Ability to export at any clause?~~

Saving power?

~~Are we working on 32 vertices at a time or 16?~~

Size of the fifo containing the information of a vector of pixels/vertices. And size of the fifos before the reservation stations.

Sequencer Instruction memory, and constant memory.

Arbitration policy for the output file.

Loops and branches.

The parameter cache may end up in the PA rather than in the RS. Parameter cache management thus may change.

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	14 August, 2001~~14~~ August, 20017 May	4 September, 2015~~7~~ September, 20016	GEN-CXXXXX-REVA	1 of 26

Author: Laurent Lefebvre

Issue To:	**Copy No:**

R400 Sequencer Specification

SEQ

Version 0.5~~2~~

Overview: This is an architectural specification for the R400 Sequencer block (SEQ). It provides an overview of the required capabilities and expected uses of the block. It also describes the block interfaces, internal sub-blocks, and provides internal state diagrams.

AUTOMATICALLY UPDATED FIELDS:

Document Location: C:\perforce\r400\arch\doc\gfx\RE\R400_Sequencer.doc

Current Intranet Search Title: R400 Sequencer Specification

APPROVALS	
Name/Dept	Signature/Date

Remarks:

THIS DOCUMENT CONTAINS CONFIDENTIAL INFORMATION THAT COULD BE SUBSTANTIALLY DETRIMENTAL TO THE INTEREST OF ATI TECHNOLOGIES INC. THROUGH UNAUTHORIZED USE OR DISCLOSURE.

Table Of Contents

1. OVERVIEW....................................... 4~~3~~
1.1 Top Level Block Diagram.................. 5~~4~~
1.2 Data Flow graph................................ 9~~7~~
1.3 Control Graph................................ 12~~10~~
2. INTERPOLATED DATA BUS....... 12~~10~~
3. INSTRUCTION STORE .... 12~~10~~
4. CONSTANT STORE..................... 13~~11~~
5. LOOPING AND BRANCHES........ 13~~11~~
6. REGISTER FILE ALLOCATION... 13~~11~~
7. TEXTURE ARBITRATION............ 14~~12~~
8. ALU ARBITRATION .... 14~~12~~
9. HANDLING STALLS .... 15~~13~~
10. CONTENT OF THE RESERVATION STATION FIFOS.................................. 15~~13~~
11. THE OUTPUT FILE (RB FIFO AND PARAMETER CACHE).......................... 15~~13~~
12. INTERFACES....................... 15~~13~~
12.1 External Interfaces............ 15~~13~~
12.1.1 Sequencer to Shader Engine Bus....................................... 15~~13~~
12.1.2 Shader Engine to Output File 15~~13~~
12.1.3 Shader Engine to Texture Unit Bus (Fast Bus) .... 16~~14~~
12.1.4 Sequencer to Texture Unit bus (Slow Bus) 16~~14~~
12.1.5 Shader Engine to RE/PA Bus 16~~14~~
12.1.6 PA? to sequencer.............. 16~~14~~
13. EXAMPLES OF PROGRAM EXECUTIONS...................................... 17~~14~~
13.1.1 Sequencer Control of a Vector of Vertices 17~~14~~
13.1.2 Sequencer Control of a Vector of Pixels 18~~16~~
13.1.3 Notes................................ 19~~17~~
14. OPEN ISSUES............................ 26~~17~~
~~1. OVERVIEW....................................... 3~~
~~1.1 Top Level Block Diagram.................... 4~~
~~1.2 Data Flow graph.................................. 7~~
~~1.3 Control Graph.................................... 10~~
~~2. INTERPOLATED DATA BUS........... 10~~
~~3. INSTRUCTION STORE .... 10~~
~~4. CONSTANT STORE.......................... 11~~
~~5. LOOPING AND BRANCHES............ 11~~
~~6. REGISTER FILE ALLOCATION....... 11~~
~~7. TEXTURE ARBITRATION................ 12~~
~~8. ALU ARBITRATION .... 12~~
~~9. HANDLING STALLS .... 13~~
~~10. CONTENT OF THE RESERVATION STATION FIFOS.. 13~~
~~11. THE OUTPUT FILE (RB FIFO AND PARAMETER CACHE)............................ 13~~
~~12. INTERFACES.............................. 13~~
~~12.1 External Interfaces................. 13~~
~~12.1.1 Sequencer to Shader Engine Bus... 13~~
~~12.1.2 Shader Engine to Output File 13~~
~~12.1.3 Shader Engine to Texture Unit Bus (Fast Bus) .... 14~~
~~12.1.4 Sequencer to Texture Unit bus (Slow Bus) 14~~
~~12.1.5 Shader Engine to RE/PA Bus 14~~
~~12.1.6 PA to sequencer........................ 14~~
~~13. OPEN ISSUES.................................. 14~~
~~1. OVERVIEW.. 3~~
~~1.1 Top Level Block Diagram.................... 4~~
~~2. TEXTURE ARBITRATION.................. 9~~
~~3. ALU ARBITRATION .... 9~~
~~4. HANDLING STALLS .... 10~~
~~5. CONTENT OF THE RESERVATION STATION FIFOS.. 10~~
~~6. INTERFACES.............................. 10~~
~~6.1 External Interfaces................... 10~~
~~6.1.1 Sequencer to Shader Engine Bus 10~~
~~6.1.2 Shader Engine to Output File 10~~
~~6.1.3 Shader Engine to Texture Unit Bus (Fast Bus)... 10~~

~~6.1.4 Sequencer to Texture Unit bus (Slow Bus) .... 11~~
~~6.1.5 Shader Engine to RE/PA Bus .. 11~~
~~7. OPEN ISSUES .... 11~~

Revision Changes:

Rev 0.1 (Laurent Lefebvre) Date: May 7, 2001	First draft.
Rev 0.2 (Laurent Lefebvre) Date : July 9, 2001	Changed the interfaces to reflect the changes in the SP. Added some details in the arbitration section.
Rev 0.3 (Laurent Lefebvre) Date : August 6, 2001	Reviewed the Sequencer spec after the meeting on August 3, 2001.
Rev 0.4 (Laurent Lefebvre) Date : August 24, 2001	Added the dynamic allocation method for register file and an example (written in part by Vic) of the flow of pixels/vertices in the sequencer.
Rev 0.4 (Laurent Lefebvre) Date : September 7, 2001	Added timing diagrams (Vic)

1. Overview

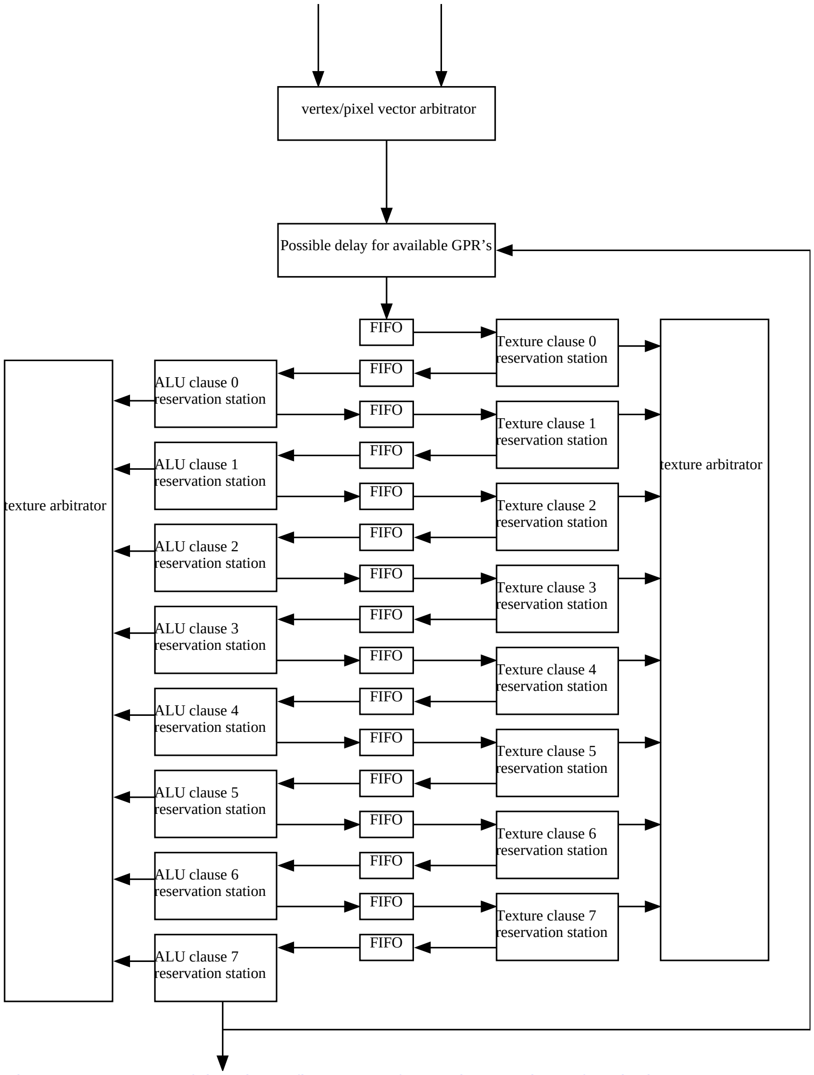
The sequencer first arbitrates between vectors of 16~~ (maybe 32)~~ vertices that arrive directly from primitive assembly and vectors of ~~8~~4 ~~-~~quads (16 pixels) ~~(32 pixels) ~~that are generated in the raster engine.

The vertex or pixel program specifies how many GPR's it needs to execute. The sequencer will not start the next vector until the needed space is available.

The sequencer is based on the R300 design. It chooses ~~an~~ two ALU clauses and a texture clause to execute, and executes all of the instructions in ~~a~~a clause before looking for a new clause of the same type. Two ALU clauses are executed interleaved to hide the ALU latency. Each vector will have eight texture and eight ALU clauses, but clauses do not need to contain instructions. A vector of pixels or vertices ping-pongs along the sequencer FIFO, bouncing from texture reservation station to alu reservation station. A FIFO exists between each reservation stage, holding up vectors until the vector currently occupying a reservation station has left. A vector at a reservation station can be chosen to execute. The sequencer looks at all eight alu reservation stations to choose an alu clause to execute and all eight texture stations to choose a texture clause to execute. The arbitrator will give priority to clauses/reservation stations closer to the ~~top~~ bottom of the pipeline. It will not execute an alu clause until the texture fetches initiated by the previous texture clause have completed. There are two separate sets of reservation stations, one for pixel vectors and one for vertices vectors. This way a pixel can pass a vertex and a vertex can pass a pixel.

To support the shader pipe the raster engine also contains the shader instruction cache and constant store. There are only one constant store for the whole chip and one instruction store. These will be shared among the four shader pipes.

1.1 Top Level Block Diagram


vertex/pixel vector arbitrator
Possible delay for available GPR's
FIFO
Texture clause 0 reservation station
Texture clause 1 reservation station
Texture clause 2 reservation station
Texture clause 3 reservation station
Texture clause 4 reservation station
Texture clause 5 reservation station
Texture clause 6 reservation station
Texture clause 7 reservation station
texture arbitrator
ALU clause 0 reservation station
ALU clause 1 reservation station
ALU clause 2 reservation station
ALU clause 3 reservation station
ALU clause 4 reservation station
ALU clause 5 reservation station
ALU clause 6 reservation station
ALU clause 7 reservation station
texture arbitrator

There are two sets of the above figure, one for vertices and one for pixels.

The rasterizer always checks the vertices FIFO first and if allowed by the sequencer sends the data to the shader. If the vertex FIFO is empty then, the rasterizer takes the first entry of the pixel FIFO (a vector of ~~32~~ 16 pixels) and sends it to the interpolators. Then the sequencer takes control of the packet. The packet consists of 3 bits of state, 6-7 bits for the base address of the Shader program and some information on the coverage to determine texture LOD. All other information (2x2 adresses) is put in a FIFO (one for the pixels and one for the vertices) and retrieved when the packet finishes its last clause.

On receipt of a packet, the input state machine (not pictured but just before the first FIFO) allocated enough space in the registers to store the interpolated values and temporaries. Following this, the input state machine stacks the packet in the first FIFO.

On receipt of a command, the level 0 texture machine issues a texure request and corresponding register address for the texture address (ta). A small command (tcmd) is passed to the texture system identifying the current level number (0) as well as the register ~~set being used~~write address for the texture return data. One texture request is sent every 4 clocks causing the texturing of four 2x2s worth of data (or 16 vertices). Once all the requests are sent the packet is put in FIFO 1.

Upon recept of the return data ~~(identified by the tcmd containing the level number 0), the level 0 texture machine issues a register address for the return value (td)~~, the texture unit writes the data to the register file using the write address that was provided by the level 0 texture machine and sends the clause number (0) to the level 0 texture state machine to signify that the write is done and thus the data is ready. Then, the level 0 texture machine ~~it~~ increments the counter of FIFO ~~one~~ 1 to signify to the ALU 1 that the data is ready to be processed.

On receipt of a command, the level 0 ALU machine first decrements the input FIFO counter and then issues a complete set of level 0 shader instructions. For each instruction, the state machine generates 3 source addresses, one destination address (~~2~~ 3 cycles later) and an instruction ~~id wich is used to index into the instruction store~~. Once the last instruction as been issued, the packet is put into FIFO 2. ~~Note that in the case of a pixel packet, the two vectors of 16 pixels are consecutive in order to hide the latency of the ALUs (8 cycles).~~

There will always be two active ALU clauses at any given time (and two arbitrers) ~~In this case, the instructions of a vector are interleaved with the instructions of the other vector.~~ One arbiter will arbitrate over the odd clock cycles and the other one will arbitrate over the even clock cycles. The only constraints between the two arbitrers is that they are not allowed to pick the same clause number as they other one is currently working on if the packet os of the same type.

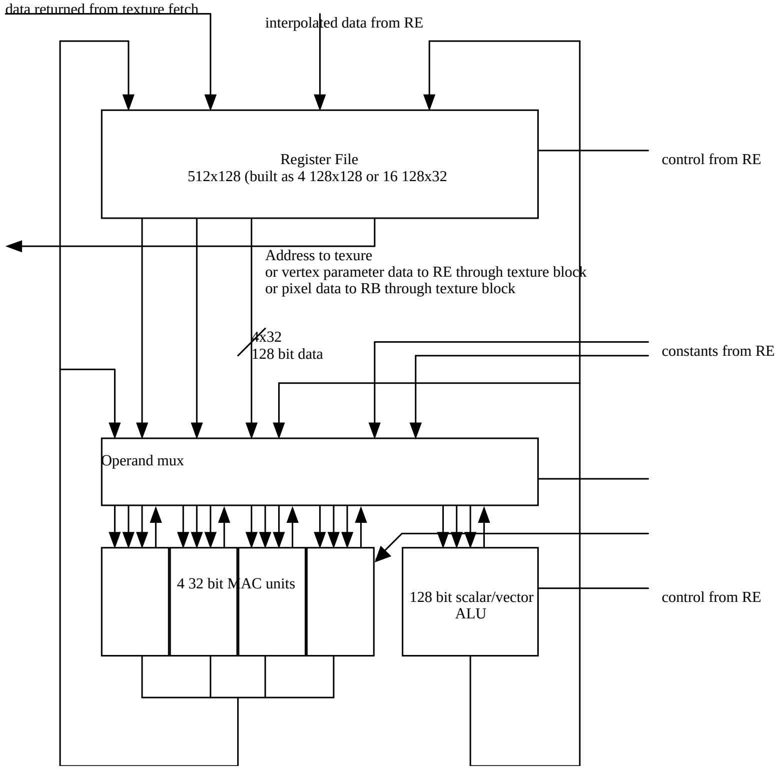
Formatted

If the packet is a vertex packet, upon reaching ALU clause 4, it can export the position if the position is ready. So the arbitrer must prevent ALU clause 4 to be selected if the positional buffer is full (or can't be accessed). Along with the positional data, the location where the vertex data is to be put is also sent (parameter data pointers).

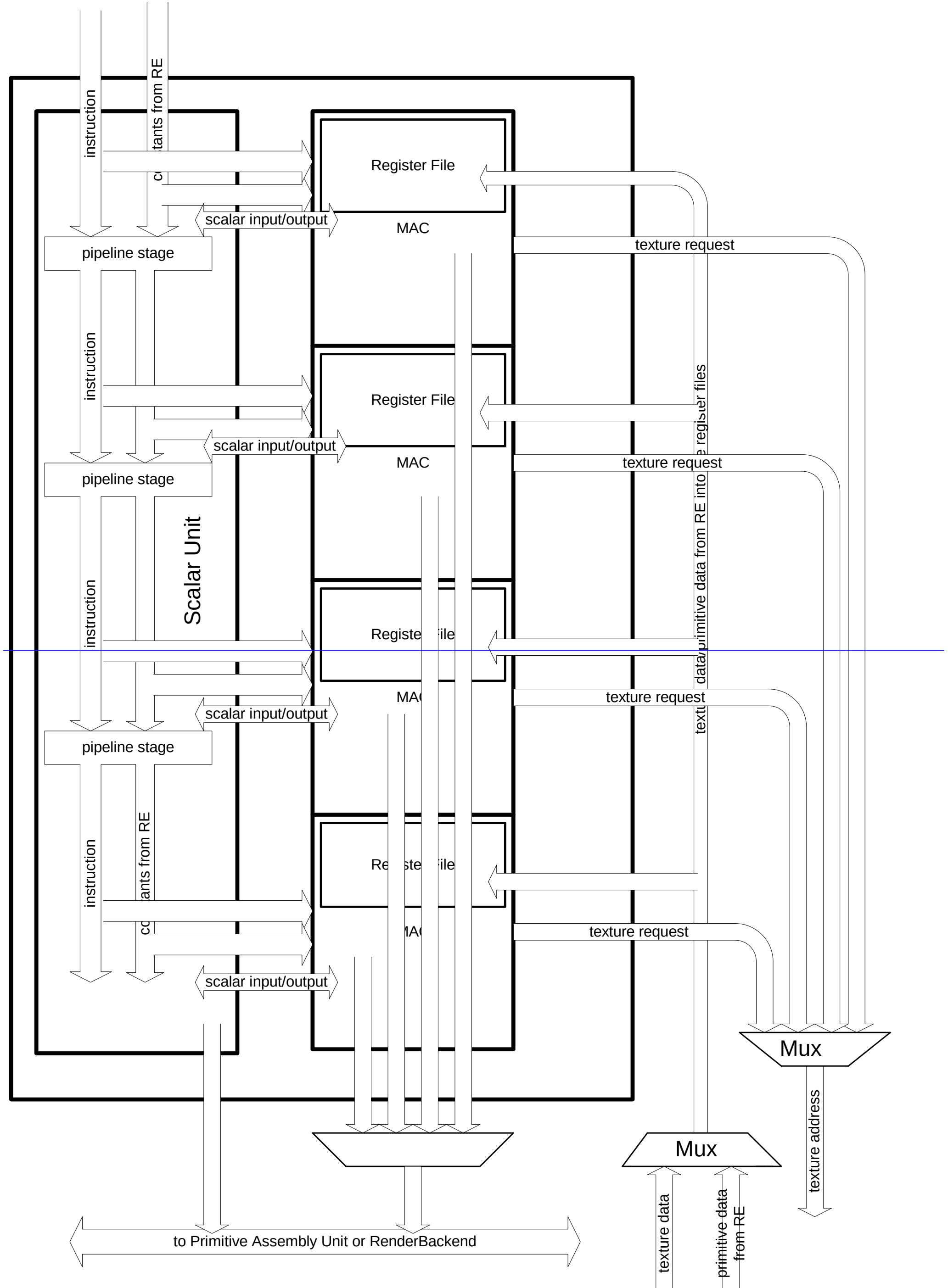
All other level process in the same way until the packet finally reaches the last ALU machine (8). On completion of the level 8 ALU clause, a valid bit is sent to the Render Backend which picks up the color data. This requires that the last instruction writes to the output register – a condition that is almost always true. If the packet was a vertex packet, instead of sending the valid bit to the RB, it is sent to the PA~~, which picks up the data and puts it into the vertex store~~ so it can know that the data present in the parameter store is valid.

Only ~~one~~ two ALU state machine may have access to the ~~SRAM~~register file address bus or the instruction decode bus at one time. Similarly, only one texture state machine may have access to the ~~SRAM~~register file address bus at one time. Arbitration is performed by ~~two~~ three arbitrer blocks (~~one~~ two for the ALU state machines and one for the texture state machines). The arbitrers always favor the higher number state machines, preventing a bunch of half finished jobs from clogging up the ~~SRAM~~register ~~S~~files.

Each state machine maintains an address pointer specifying where the 16 ~~(or 32)~~ entries vector is located in the ~~SRAM~~register file (the texture machine has two pointers one for the read address and one for the write). Upon completion of its job, the address pointer is incremented by a predefined amount equal to the total number of registers required by the shading code. A comparison of the address pointer for the first state machine in the chain (the input state machine), and the last machine in the chain (the level 8 ALU machine), gives an indication of how much unallocated ~~SRAM~~register file memory is available

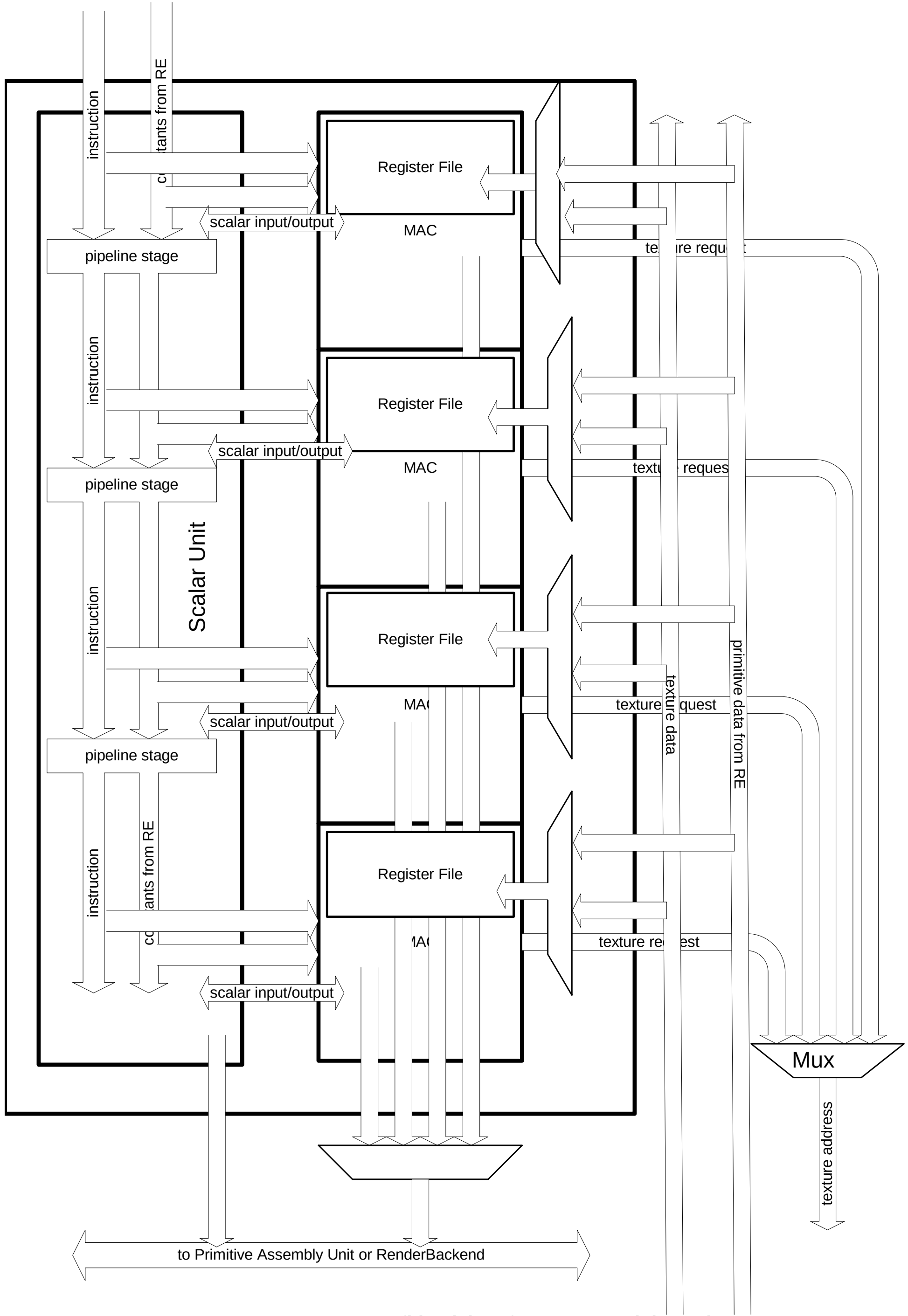


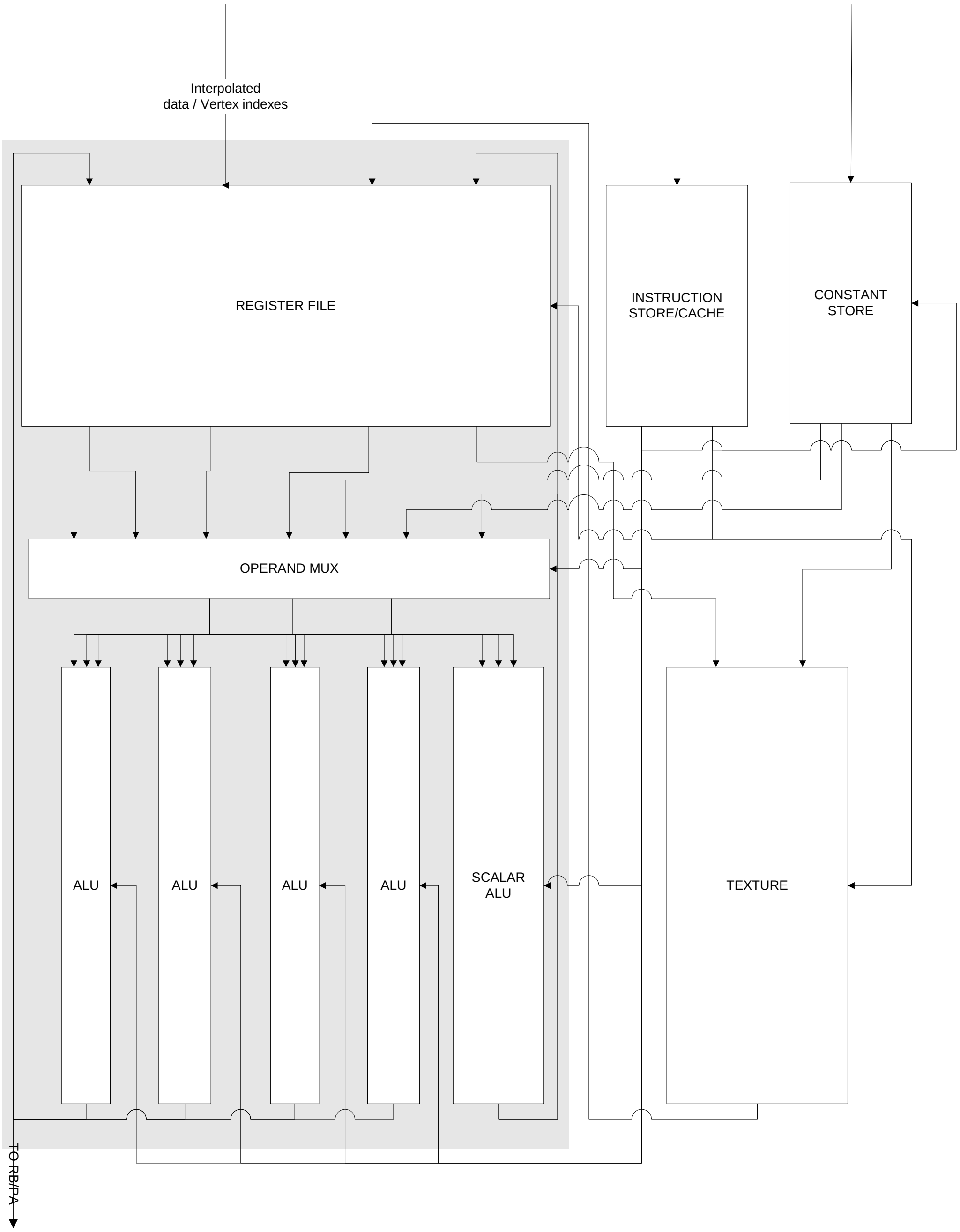
data returned from texture fetch
interpolated data from RE
Register File
512x128 (built as 4 128x128 or 16 128x32
control from RE
Address to texure
or vertex parameter data to RE through texture block
or pixel data to RB through texture block
4x32
128 bit data
constants from RE
Operand mux
4 32 bit MAC units
128 bit scalar/vector
ALU
control from RE



instruction
constants from RE
Register File
MAC
scalar input/output
pipeline stage
texture request
Scalar Unit
texture data/primitive data from RE into the register files
Mux
texture address
texture data
primitive data from RE
to Primitive Assembly Unit or RenderBackend

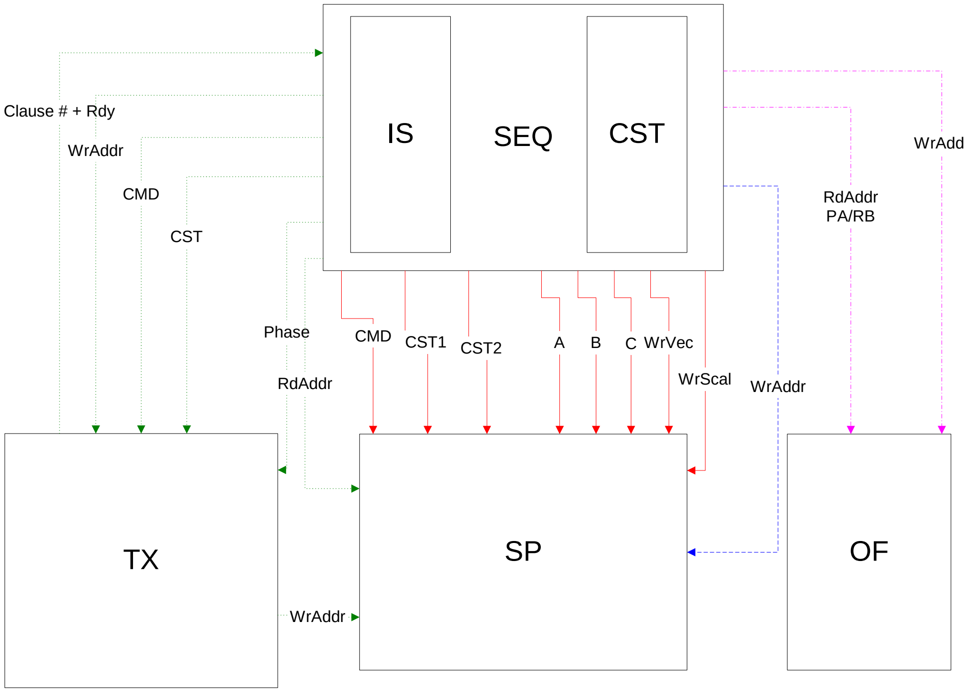
1.2 Data Flow graph


instruction
constants from RE
Register File
MAC
scalar input/output
pipeline stage
instruction
Register File
scalar input/output
MAC
texture request
pipeline stage
instruction
Scalar Unit
Register File
MAC
texture request
texture data
primitive data from RE
scalar input/output
pipeline stage
instruction
constants from RE
Register File
MAC
texture request
scalar input/output
Mux
texture address
to Primitive Assembly Unit or RenderBackend



The gray area represents blocks that are replicated 4 times per shader pipe (16 times on the overall chip).

1.3 Control Graph



In green is represented the Texture control interface, in red the ALU control interface, in blue the Interpolated/Vector control interface and in purple is the output file control interface.

2. Interpolated data bus

Formatted: Bullets and Numbering

Since each of the register file is actually physically divided (one 128x128 per MAC) and we don't have the place to hold a maximum size vector of vertices in the parameter buffer, we need to interpolate on a parameter basis rather than on a quad basis. So the order to the register file will be:

Q0P0 Q1P0 Q2P0 Q3P0 Q0P1 Q1P1 Q2P1 Q3P1 Q0P2 Q1P2 …

3. Instruction Store

Formatted: Bullets and Numbering

There is going to be only one instruction store for the whole chip. It may contain up to 2000 instructions of 96 bits each.

{ISSUE : The instruction store is loaded by the sequencer using the memory hub ?}.

The read bandwith from this store is 24 bits/clock/pipe. To achieve this this instruction store is likely to be broken up into 4 blocks. An ALU instruction section (1R/1W) split in two and a texture section (1R/1W) also split in two. The bandwith out of those memories is 96 bits/clock.

4. Constant Store

Formatted: Bullets and Numbering

The constant store is managed by the CP. The sequencer is aware of where the constants are using a remaping table also managed by the CP. A likely size for the constant store is 512x128 bits. The constant store is also planned to be shared. The read BW from the constant store is 512/4 bits/clock/pipe and the write bandwith is 32/4 bits/clock.

5. Looping and Branches

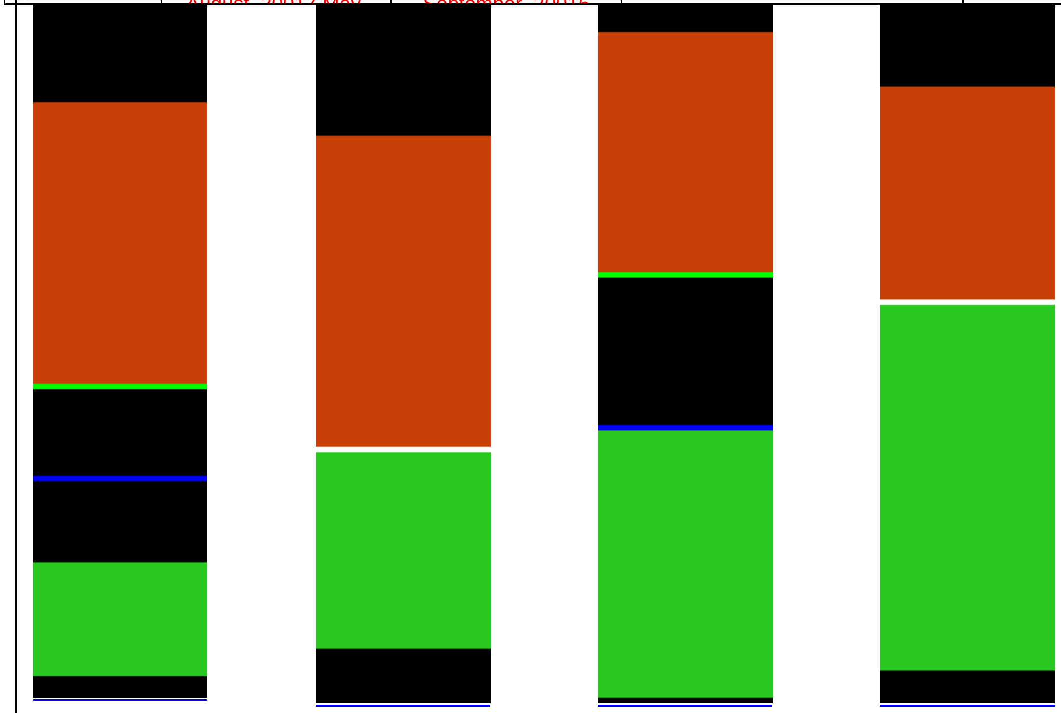
Formatted: Bullets and Numbering

Loops and branches are planned to be supported and will have to be dealt with at the sequencer level. However, it is still unclear if we plan on supporting data dependent branches or not.

6. Register file allocation

Formatted: Bullets and Numbering

The register file allocation for vertices and pixels can either be static or dynamic. In both cases, the register file in managed using two round robins (one for pixels and one for vertices). In the dynamic case the boundary between pixels and vertices is allowed to move, in the static case it is fixed to VERTEX_REG_SIZE for vertices and 256-VERTEX_REG_SIZE for pixels.



Above is an example of how the algorithm works. Vertices come in from top to bottom; pixels come in from bottom to top. Vertices are in orange and pixels in green. The blue line is the tail of the vertices and the green line is the tail of the pixels. Thus anything between the two lines is shared. When pixels meets vertices the line turns white and the boundary is static until both vertices and pixels share the same "unallocated bubble". Then the boundary as allowed moving again.

Formatted: Bullets and Numbering

~~2.~~7. Texture Arbitration

The texture arbitration logic chooses one of the 8 potentially pending texture clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute. Once chosen, the clause state machine will send one 2x2 texture fetch per clock (or 4 fetches in one clock every 4 clocks) until all the texture fetch instructions of the clause are sent. This means that there cannot be any dependencies between two texture fetches of the same clause.

The arbitrator will not wait for the texture fetches to return prior to selecting another clause for execution. The texture pipe will be able to handle up to ~~100~~X(?) in flight texture fetches and thus there can be a fair number of active clauses waiting for their texture return data.

Formatted: Bullets and Numbering

~~3.~~8. ALU Arbitration

ALU arbitration proceeds in almost the same way than texture arbitration. The ALU arbitration logic chooses one of the 8 potentially pending ALU clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute~~execute. If the packet chosen is a packet of vertices, the state machine issues one instruction every 4 clocks until the clause is finished. This means that the compiler has to insert nops between two dependent successive instructions. If the packet is a pixel packet it is made out of two sub-vectors of 16. Thus the state machine issues the first instruction for the first sub-vector and then, 4 clocks later, the first instruction of the second sub-vector and so on until the clause is finished. ~~. There are two ALU arbitrers, one for the even clocks and

one for the odd clocks. For exemple, here is the sequencing of two interleaved ALU clauses (E and O stands for Even and Odd):

Einst0 Oinst0 Einst1 Oinst1 Einst2 Oinst2 Einst0 Oinst3 Einst1 Oinst4 Einst2 Oinst0…
Proceeding this way hides the latency of 8 clocks of the ALUs.

~~4.~~9. Handling Stalls

Formatted: Bullets and Numbering

When the output file is full, the sequencer prevents the ALU arbitration logic to select the last clause (this way nothing can exit the shader pipe until there is place in the output file. If ~~we have the ability to export at any clause~~the packet is a vertex packet and the position buffer is full (POS_FULL) then the sequencer also prevents a thread to enter the exporting clause (4?). The sequencer will set the OUT_FILE_FULL signal n clocks before the output file is actually full and thus the ALU arbitrer will be able read this signal and act accordingly by not preventing exporting clauses to proceed.

~~5.~~10. Content of the reservation station FIFOs

Formatted: Bullets and Numbering

3 bits of Render State ~~and~~ 6-7 bits for the base address of the instruction store and some bits for LOD correction. Every other information (such as the coverage mask, quad address, etc.) is put in a FIFO and is retrieved when the quad exits the shader pipe to enter in the output file buffer. Since pixels and vertices are kept in order in the shader pipe, we only need two fifos (one for vertices and one for pixels) deep enough to cover the shader pipe latency. This size will be determined later when we will know the size of the small fifos between the reservation stations.

~~6.~~11. The Output File (RB FIFO and Parameter Cache)

Formatted: Bullets and Numbering

The output file is where program results are exported when the pixel/vertex shader finishes. It constists of a 512x128 memory cell that is statically divided between pixels and vertices. ~~Each section is a regular FIFO.~~ The output file has 1 write port and 1 read port. The sequencer is responsible for managing the addresses of this output file and for stalling the shader pipe should this output file fill up. The management is done by keeping the tail and head pointers of each sections (pixels and vertices) and incrementing them using a simple RoundRobin allocation policy. The sequencer must also arbitrate between the PA and the RB for the use of the read port. This arbitration will either be priority based or just interleaved evenly (1 read every 2 clocks for each of the blocks).

~~7.~~12. Interfaces

Formatted: Bullets and Numbering

~~7.1~~12.1 External Interfaces

~~7.1.1~~12.1.1 *Sequencer to Shader Engine Bus*

This is a bus that sends the instruction and constant data to all 4 Sub-Engines of the Shader. Because a new instruction is needed only every 4 clocks, the width of the bus is divided by 4 and both constants and instruction are sent over those 4 clocks.

Name	Direction	Bits	Description
Instruction Start	SEQ-> SP	1	High on first cycle of transfer
Constant 0	SEQ-> SP	32	128 bits transferred over 4 cycles, alpha first…blue last
Constant 1	SEQ-> SP	32	128 bits transferred over 4 cycles, alpha first…blue last
Instruction	SEQ-> SP	30	120 bits transferred over 4 cycles (order TBD) ?

~~7.1.2~~12.1.2 *Shader Engine to Output File*

Formatted: Bullets and Numbering

Every clock each Sub-Engine can output 128 bits of 'vector' data and 32 bits of 'scalar' data to an output file (?). This data will be compressed into 128 bits total prior to storage in output file.

Name	Direction	Bits	Description
UL_Vector_Out	SP-> OF	128	Vector Data out

UL_Scalar_Out	SP-> OF	32	Vector Data out
UR_Vector_Out	SP-> OF	128	Vector Data out
UR_Scalar_Out	SP-> OF	32	Vector Data out

Name	Direction	Bits	Description
LL_Vector_Out	SP-> OF	128	Vector Data out
LL_Scalar_Out	SP-> OF	32	Vector Data out
LR_Vector_Out	SP-> OF	128	Vector Data out
LR_Scalar_Out	SP-> OF	32	Vector Data out

~~7.1.3~~12.1.3 *Shader Engine to Texture Unit Bus (Fast Bus)*

Formatted: Bullets and Numbering

One quad's worth of addresses is transferred to Texture Unit every clock. These are sourced from a different pixel within each of the sub-engines repeating every 4 clocks. The ~~register file~~register file index to read must precede the data by 2 clocks. The Read address associated with Quad 0 must be sent 1 clock after the Instruction Start signal is sent, so that data is read 3 clocks after the Instruction Start.

One Quad's worth of Texture Data may be written to the ~~Register File~~Register file every clock. These are directed to a different pixel of the sub-engines repeating every 4 clocks. The ~~register file~~register file index to write must accompany the data. Data and Index associated with the Quad 0 must be sent 3 clocks after the Instruction Start signal is sent.

Name	Direction	Bits	Description
Tex_Read_Register_Index	SEQ->SP	8	Index into ~~Register File~~Register files for reading Texture Address
Tex_RegFile_Read_Data	SP->TEX	512	4 Texture Addresses read from the ~~Register File~~Register file
Tex_Write_Register_Index	SEQ->TEX	8	Index into ~~Register file~~Register file for write of returned Texture Data

~~7.1.4~~12.1.4 *Sequencer to Texture Unit bus (Slow Bus)*

Formatted: Bullets and Numbering

Once every four clock, the texture unit sends to the sequencer on wich clause it is now working and if the data in the registers is ready or not. This way the sequencer can update the texture counters for the reservation station fifos. The sequencer also provides the intruction and constants for the texture fetch to execute and the address in the ~~register file~~register file where to write the texture return data.

Name	Direction	Bits	Description
Tex_Ready	TEX→ SEQ	1	Data ready
Tex_Clause_Num	TEX→ SEQ	3	Clause number
Tex_cst	SEQ→TEX	?	Texture constants X bits sent over 4 clocks
Tex_Inst	SEQ→TEX	?	Texture fetch instruction X bits sent over 4 clocks

~~7.1.5~~12.1.5 *Shader Engine to RE/PA Bus*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
Interpolator_Register_Index	SEQ->SP	8	Index into ~~Register File~~Register files for write of Interpolator/Index Data
Interpolator_Write_Mask	SEQ->SP	1	Write Mask. The same write mask is used for all 4 pixels
Interpolator_Write_Data	RE/PA->SP	512	4 interpolated vectors or vectors of indices

12.1.6 *PA? to sequencer*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
Adress	PA→SEQ	?	Dealocation adress sent by the PA telling the Sequencer that it is now possible to free this space in the parameter buffer. This token is a pointer in the parameter cache and 4 bits to tell the size wich is to be freed up.

Formatted

Formatted

Formatted

Formatted

Formatted: Bullets and Numbering

13. Examples of program executions

13.1.1 *Sequencer Control of a Vector of Vertices*

1. PA sends a vector of 16 vertices (actually vertex indices – 32 bits/index for 512 bit total) to the RE's Vertex FIFO
   - state pointer as well as tag into position cache is sent along with vertices
   - space was allocated in the position cache for transformed position before the vector was sent
   - **also before the vector is sent to the RE, the CP has loaded the global instruction store with the vertex shader program (using the MH?)**
   - The vertex program is assumed to be loaded when we receive the vertex vector.
     - the SEQ then accesses the IS base for this shader using the local state pointer (provided to all sequencers by the RBBM when the CP is done loading the program)
2. SEQ arbitrates between the Pixel FIFO and the Vertex FIFO – basically the Vertex FIFO always has priority
   - at this point the vector is removed from the Vertex FIFO
   - the arbitrer is not going to select a vector to be transformed if the parameter cache is full unless the pipe as nothing else to do (ie no pixels are in the pixel fifo).
3. SEQ allocates space in the SP register file for index data plus GPRs used by the program
   - the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer that came down with the vertices
   - SEQ will not send vertex data until space in the register file has been allocated
4. SEQ sends the vector to the SP register file over the RE_SP interface (which has a bandwidth of 512 bits/cycle)
   - the 16 vertex indices are sent to the 16 register files over 4 cycles
     - RF0 of SU0, SU1, SU2, and SU3 is written the first cycle
     - RF1 of SU0, SU1, SU2, and SU3 is written the second cycle
     - RF2 of SU0, SU1, SU2, and SU3 is written the third cycle
     - RF3 of SU0, SU1, SU2, and SU3 is written the fourth cycle
   - the index is written to the least significant 32 bits **(floating point format?) (what about compound indices)** of the 128-bit location within the register file (w); the remaining data bits are set to zero (x, y, z)
5. SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of texture state machine 0, or TSM0 FIFO)
   - the control packet contains the state pointer, the tag to the position cache and a register file base pointer.
6. TSM0 accepts the control packet and fetches the instructions for texture clause 0 from the global instruction store
   - TSM0 was first selected by the TSM arbiter before it could start
7. all instructions of texture clause 0 are issued by TSM0
8. the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)
   - TSM0 does not wait for requests made to the Texture Unit to complete; it passes the register file write index for the texture data to the TU, which will write the data to the RF as it is received
   - once the TU has written all the data to the register files, it increments a counter that is associated with ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go ahead start to execute the ALU clause
9. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store
10. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of texture state machine 1, or TSM1 FIFO)
11. the control packet continues to travel down the path of reservation stations until all clauses have been executed
    - position can be exported in ALU clause 3 (or 4?); the data (and the tag) is sent over a position bus (which is shared with all four shader pipes) back to the PA's position cache
    - A parameter cache pointer is also sent along with the position data. This tells to the PA where the data is going to be in the parameter cache.
      - there is a position export FIFO in the SP that buffers position data before it gets sent back to the PA

    - the ASM arbiter will prevent a packet from starting an exporting clause if the position export FIFO is full
- parameter data is exported in clause 7 (as well as position data if it was not exported earlier)
    - parameter data is sent to the Parameter Cache over a dedicated bus
    - the SEQ allocates storage in the Parameter Cache, and the SEQ deallocates that space when there is no longer a need for the parameters (it is told by the PA when using a token).
    - the ASM arbiter will prevent a packet from starting on ASM7 if the parameter cache (or the position buffer if position is being exported) is full

12. after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program

13.1.2 *Sequencer Control of a Vector of Pixels*

1. **As with vertex shader programs, pixel shaders are loaded into the global instruction store by the CP**
    - At this point it is assumed that the pixel program is loaded into the instruction store and thus ready to be read.
2. the ~~RE's~~ Parameter Buffer is loaded from the Parameter Cache before the SEQ takes control of the vector
    - after the HZ culling stage a request is made by the RE to send parameter data to the Parameter buffer
    - the Parameter buffer is wide enough to source 3 vertices worth of a particular parameter in one cycle
    - **at this moment the right sequencer will free up the parameter store locations not used anymore using the token provided by the PA**.
3. the RE's Pixel FIFO is loaded with the barycentric coordinates for pixel quads by the detailed walker
    - the state pointer and the LOD correction bits are also placed in the Pixel FIF0
    - the Pixel FIFO is wide enough to source one quad's worth of barycentrics per cycle
4. SEQ arbitrates between Pixel FIFO and Vertex FIFO – when there are no vertices pending OR there is no space left in the register files for vertices, the Pixel FIFO is selected
5. SEQ allocates space in the SP register file for all the GPRs used by the program
    - the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer
    - SEQ will not allow interpolated data to be sent to the shader until space in the register file has been allocated
6. SEQ control starts with the interpolation of parameters (up to 16 per thread) by sending the barycentric coordinates from the Pixel FIFO and the parameters from the Parameter Buffer to the interpolator
    - P0i, P0j, and P0k (the value of P0 at each vertex) are loaded into the interpolator from the Parameter buffer
    - Q0 i, j, and k are loaded into the interpolator from the Pixel FIFO
    - The interpolator then generates the parameter value for each pixel in Q0 (Q0P0)
    - **P0i, P0j, and P0k are sent to the interpolator for Q1 only if Q1 is from a different primitive; if Q1 is from the same primitive as Q0, then the P0i, P0j, and P0k values loaded for Q0 are held by the interpolator and reused for Q1**
        - **a "different_prim" control bit is passed with the barycentric data for each quad in the Pixel FIFO that indicates whether new parameter data needs to be loaded into the interpolator**
    - Q1 i, j, and k are then loaded into the interpolator from the Pixel FIFO
    - The interpolator then generates the parameter value for each pixel in Q1 (Q1P0)
    - Q2P0 and Q3P0 are generated in a similar manner
    - The next set of parameter data - P1i, P1j, and P1k - is then loaded into the interpolator
    - Q0 i, j, and k now must be re-read from the Pixel FIFO – this means that the output of the Pixel FIFO loops through the top four entries on each read command until at the end a final "block_pop" signal is asserted, causing the top four sets of barycentric coordinates to finally be removed
    - so the order of parameter info generated is Q0P0, Q1P0, Q2P0, Q3P0, Q0P1, Q1P1, etc.
7. SEQ controls the transfer of interpolated data to the SP register file over the RE_SP interface (which has a bandwidth of 512 bits/cycle)
    - 16 pixels worth of interpolated parameter data is sent to the 16 register files over 4 cycles
        - RF0 of SU0, SU1, SU2, and SU3 is written with Q0P0 the first cycle
        - RF1 of SU0, SU1, SU2, and SU3 is written with Q1P0 second cycle
        - RF2 of SU0, SU1, SU2, and SU3 is written with Q2P0 third cycle
        - RF3 of SU0, SU1, SU2, and SU3 is written with Q3P0 fourth cycle

8. SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of texture state machine 0, or TSM0 FIFO)
    - note that there is a separate set of reservation stations/arbiters/state machines for vertices and for pixels
    - the control packet contains the state pointer, the register file base pointer, and the LOD correction bits
    - all other informations (such as quad address for example) travels in a separate FIFO

9. TSM0 accepts the control packet and fetches the instructions for texture clause 0 from the global instruction store
    - TSM0 was first selected by the TSM arbiter before it could start

10. all instructions of texture clause 0 are issued by TSM0

11. the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)
    - TSM0 does not wait for texture requests made to the Texture Unit to complete; it passes the register file write index for the texture data to the TU, which will write the data to the RF as it is received
    - once the TU has written all the data for a particular clause to the register files, it increments a counter that is associated with the ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go ahead and pop the FIFO and start to execute the ALU clause

12. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store

13. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of texture state machine 1, or TSM1 FIFO)

14. the control packet continues to travel down the path of reservation stations until all clauses have been executed
    - pixel data is exported in the last ALU clause (clause 7)
        - it is sent to an output FIFO where it will be picked up by the render backend
        - the ASM arbiter will prevent a packet from starting on ASM7 if the output FIFO is full

15. after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program

13.1.3 *Notes*

Formatted: Bullets and Numbering

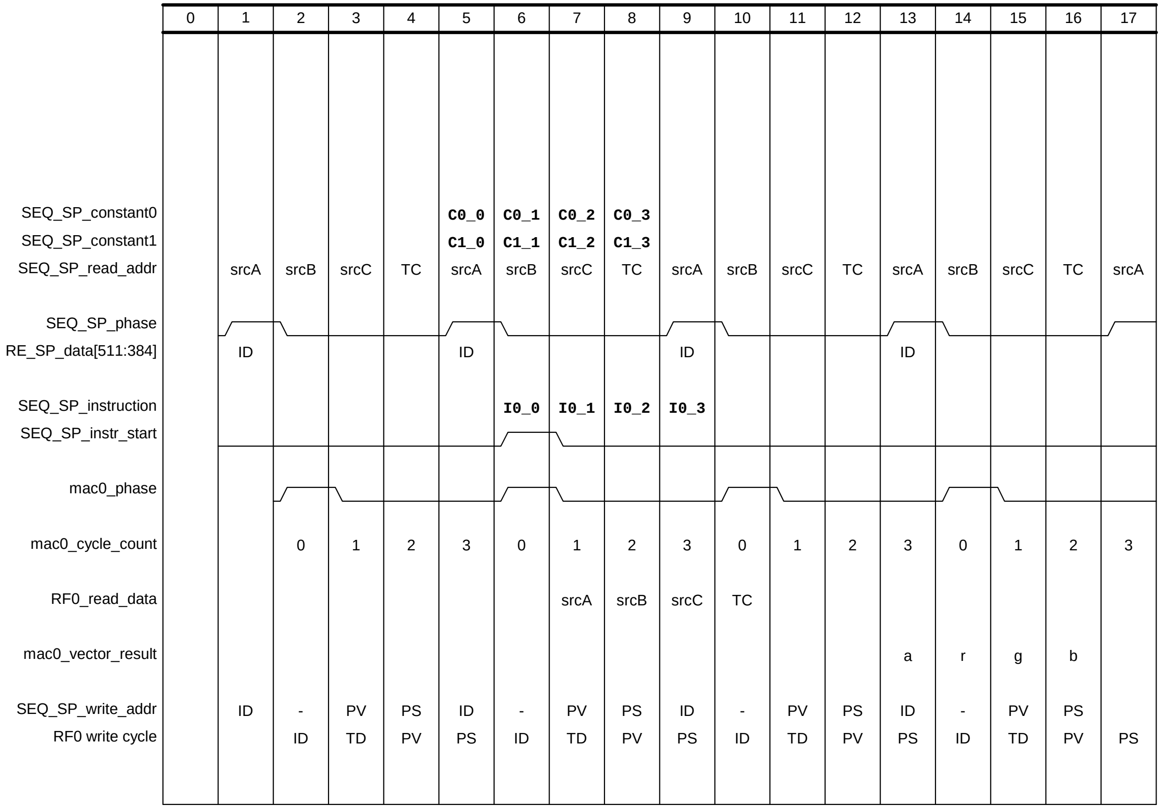
16. the state machines and arbitrers will operate ahead of time so that they will be able to immediately start the real threads or stall.

17. the register file base pointer for a vector needs to travel with the vector through the reservation stations, but the instruction store base pointer does not – this is because the RF pointer is different for all threads, but the IS pointer is only different for each state and thus can be accessed via the state pointer

18. Waterfalling, parameter buffer allocation, loops and branches and parameter cache de-allocation still needs to be specked out.

14. Timing Diagrams

14.1 MAC 0

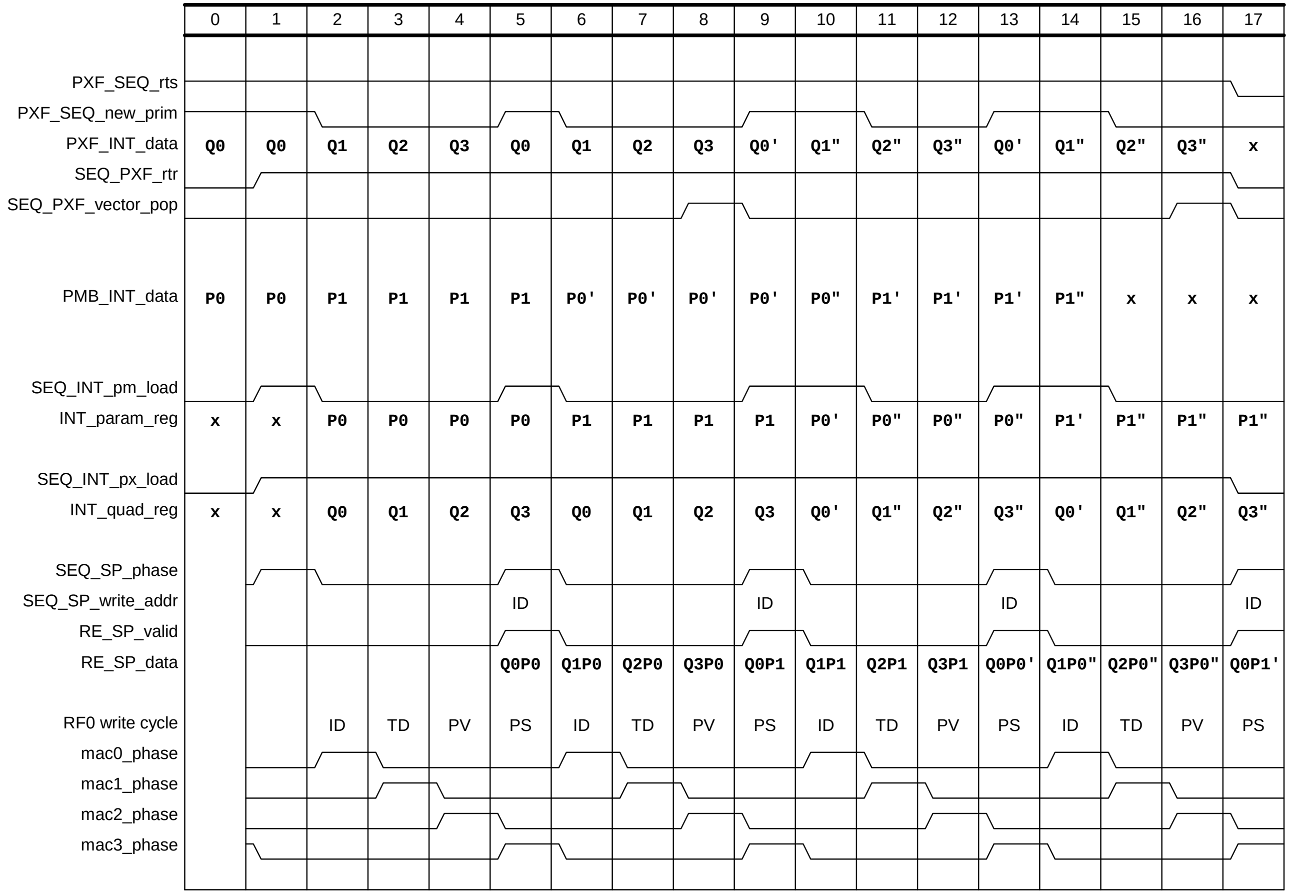


	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
SEQ_SP_constant0						C0_0	C0_1	C0_2	C0_3									
SEQ_SP_constant1						C1_0	C1_1	C1_2	C1_3									
SEQ_SP_read_addr		srcA	srcB	srcC	TC	srcA	srcB	srcC	TC	srcA	srcB	srcC	TC	srcA	srcB	srcC	TC	srcA
SEQ_SP_phase																		
RE_SP_data[511:384]		ID				ID				ID				ID				
SEQ_SP_instruction							I0_0	I0_1	I0_2	I0_3								
SEQ_SP_instr_start																		
mac0_phase																		
mac0_cycle_count			0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
RF0_read_data								srcA	srcB	srcC	TC							
mac0_vector_result														a	r	g	b	
SEQ_SP_write_addr		ID	-	PV	PS	ID	-	PV	PS	ID	-	PV	PS	ID	-	PV	PS	
RF0 write cycle			ID	TD	PV	PS	ID	TD	PV	PS	ID	TD	PV	PS	ID	TD	PV	PS

Timing Diagram 1: Sequencer to Shader Pipe 0, Shader Unit 0, MAC 0

14.2 Sequencer to Shader Pipe

Formatted: Bullets and Numbering

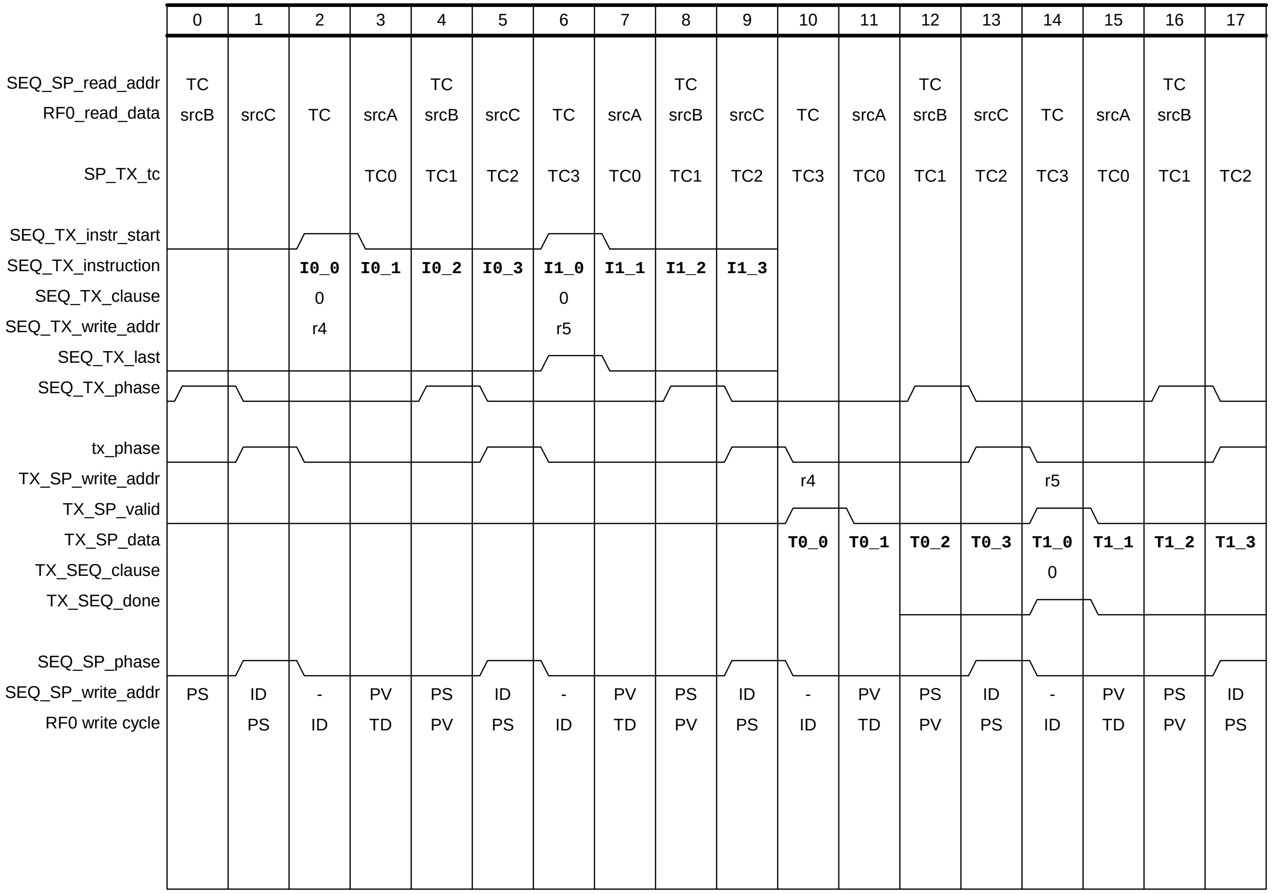


	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
PXF_SEQ_rts																		
PXF_SEQ_new_prim																		
PXF_INT_data	Q0	Q0	Q1	Q2	Q3	Q0	Q1	Q2	Q3	Q0'	Q1"	Q2"	Q3"	Q0'	Q1"	Q2"	Q3"	x
SEQ_PXF_rtr																		
SEQ_PXF_vector_pop																		
PMB_INT_data	P0	P0	P1	P1	P1	P1	P0'	P0'	P0'	P0'	P0"	P1'	P1'	P1'	P1"	x	x	x
SEQ_INT_pm_load																		
INT_param_reg	x	x	P0	P0	P0	P0	P1	P1	P1	P1	P0'	P0"	P0"	P0"	P1'	P1"	P1"	P1"
SEQ_INT_px_load																		
INT_quad_reg	x	x	Q0	Q1	Q2	Q3	Q0	Q1	Q2	Q3	Q0'	Q1"	Q2"	Q3"	Q0'	Q1"	Q2"	Q3"
SEQ_SP_phase																		
SEQ_SP_write_addr						ID				ID				ID				ID
RE_SP_valid																		
RE_SP_data						Q0P0	Q1P0	Q2P0	Q3P0	Q0P1	Q1P1	Q2P1	Q3P1	Q0P0'	Q1P0"	Q2P0"	Q3P0"	Q0P1'
RF0 write cycle			ID	TD	PV	PS	ID	TD	PV	PS	ID	TD	PV	PS	ID	TD	PV	PS
mac0_phase																		
mac1_phase																		
mac2_phase																		
mac3_phase																		

Timing Diagram 2: RE Interpolator to Shader Pipe Data Transfer

14.3 Sequencer to Texture Pipe

Formatted: Bullets and Numbering



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
SEQ_SP_read_addr	TC				TC				TC				TC				TC	
RF0_read_data	srcB	srcC	TC	srcA	srcB	srcC	TC	srcA	srcB	srcC	TC	srcA	srcB	srcC	TC	srcA	srcB	
SP_TX_tc				TC0	TC1	TC2	TC3	TC0	TC1	TC2	TC3	TC0	TC1	TC2	TC3	TC0	TC1	TC2
SEQ_TX_instr_start																		
SEQ_TX_instruction			I0_0	I0_1	I0_2	I0_3	I1_0	I1_1	I1_2	I1_3								
SEQ_TX_clause			0				0											
SEQ_TX_write_addr			r4				r5											
SEQ_TX_last																		
SEQ_TX_phase																		
tx_phase																		
TX_SP_write_addr											r4				r5			
TX_SP_valid																		
TX_SP_data											T0_0	T0_1	T0_2	T0_3	T1_0	T1_1	T1_2	T1_3
TX_SEQ_clause															0			
TX_SEQ_done																		
SEQ_SP_phase																		
SEQ_SP_write_addr	PS	ID	-	PV	PS	ID	-	PV	PS	ID	-	PV	PS	ID	-	PV	PS	ID
RF0 write cycle		PS	ID	TD	PV	PS	ID	TD	PV	PS	ID	TD	PV	PS	ID	TD	PV	PS

Timing Diagram 3: Sequencer - Texture Unit Interface and Texture Unit - Shader Pipe Data Transfer

14.4 Timing diagrams explanations

Formatted: Bullets and Numbering

The numbering of the four shader pipes, the four shader units, and the four MACs is from left to right and from 0 to 3. So for example the most significant 512 bits of a SP goes to SU0 and the least significant 512 bits go to SU3; within SU0, the most significant 128 bits go to MAC0 and the least significant 128 bits go to MAC3.
The following assumptions are made:

1. all block to block signals are register to register
2. for register file reads, the RF read data is available in the MAC one clock after a RF read address is registered into the MAC (this is the same as saying the read data is valid out of the RF two clocks after the address is asserted on the SEQ to SP interface)

Formatted: Bullets and Numbering

14.4.1 *Timing Diagram 1: Sequencer to Shader Pipe 0, Shader Unit 0, MAC 0*

This diagram shows the basics of the Sequencer to Shader Pipe interface. For simplicity only the timing relative to MAC0 is shown. The timing for MAC1 is one clock later than MAC0, MAC2 one clock later than MAC1, etc. This means that most of the signals need to be delayed in the SP by one cycle for MAC1, two cycles for MAC2, and three cycles for MAC3.
SEQ_SP_constant0: Constant 0 (128 bits over 4 cycles). Pipelined in SP for other MACs.
SEQ_SP_constant1: Constant 1 (128 bits over 4 cycles). Pipelined in SP for other MACs.
SEQ_SP_read_addr: Register File Read Address (8 bits). Pipelined in SP for other MACs.
SEQ_SP_phase: This signal syncs the data transfer to the RF from the RE, as well as defining the order of all writes into the RF. It is asserted during the cycle that interpolated data (ID) is valid on the RE_SP_ID bus. Pipelined in SP for other MACs.
RE_SP_ID[511:384]: This is the most significant 128 bits of the RE_SP data interface (meaning that this MAC0 is in SU0).
SEQ_SP_instruction: 96 bits of instruction are sent over 4 cycles. Pipelined in SP for other MACs.
SEQ_SP_instr_start: control bit that signals the first cycle of the instruction transfer. Pipelined in SP for other MACs.
mac0_phase: registered version of SEQ_SP_phase used in MAC0 (this may not be he actual signal name).
mac0_cycle_count: a counter inside the MAC that keeps track of the RF write cycles; 0 here corresponds to the cycle RE interpolated data is written (this may not be he actual signal name).
RF0_read_data: data that is read out of MAC0's register file (this may not be he actual signal name).
mac0_vector_result: the 32-bit output of the vector ALU (PV is built up over 4 cycles) (this may not be he actual signal name).
SEQ_SP_write_addr: Register File Write Address (8 bits). Note that the SEQ does not send the Texture Data write address over this bus. Pipelined in SP for other MACs.
RF0 write cycle: the cycles allocated to the different write sources (ID = Interpolated Data, TD = Texture Data, PV = Previous Vector, PS = Previous Scalar) (not a signal – just a reference point on the diagram).

14.4.2 *Timing Diagram 2: RE Interpolator to Shader Pipe Data Transfer*

Formatted: Bullets and Numbering

This diagram shows how pixel data (barycentric coordinates i, j, and k) is sent from the Pixel FIFO to the interpolator under SEQ control, and how parameter data (for each vertex) is also sent to the interpolator under SEQ control. The output of the Interpolator is then shown being sent over the RE_SP interface.
PXF_SEQ_rts: Indicates that the output of the pixel FIFO is valid.
PXF_SEQ_new_prim: The current output of the Pixel FIFO is from a different primitive that the previous output. Tells the SEQ that new parameter info must be fetched (if its not from a new prim, then new parameter data is not needed).
PXF_INT_data: Data output of the Pixel FIFO – goes to the Interpolator.
SEQ_PXF_rtr: Indicates that the current Pixel FIFO output will be taken by the Interpolator (driven by SEQ). Then next quad of data will be driven the next cycle.
SEQ_PXF_vector_pop: SEQ tells the Pixel FIFO to pop a vector of pixels (otherwise RTRs cause the data to be cycled between the four quads).

PMB_INT_data: Data from the Parameter Buffer to the Interpolator. (Note that the control of the parameter buffer is TBD).

SEQ_INT_pm_load: controls the loading of parameter data into the Interpolator.

INT_param_reg: register in the Interpolator that holds the per-vertex parameter data while the per-pixel parameters are generated for one or more quads (may not be the actual signal name).
SEQ_INT_px_load: controls the loading of pixel data into the Interpolator.
INT_quad_reg: : register in the Interpolator that holds one quad's worth of pixel data(may not be the actual signal name).

SEQ_SP_phase: see above under TD1.
SEQ_SP_write_addr: see above under TD1.
RE_SP_valid: Interpolator Data Valid – indicates that the SP should write the ID on the appropriate cycle.
RE_SP_data: Data from the RE interpolator to the SP.
RF0 write cycle: see above under TD1.
mac*_phase: see above under TD1. These phase signals help to show the timing offset between the MACs. Note also that each Shader Unit has a set of these signals (all with the same timing).

14.4.3 *Timing Diagram 3: Sequencer - Texture Unit Interface and Texture Unit - Shader Pipe Data Transfer*

Formatted: Bullets and Numbering

This diagram starts with the texture coordinate read from the register file and its transfer to the TX. The instruction transfer is then shown, followed by the texture data transfer to the shader pipe.
SEQ_SP_read_addr: see above. Here shows the cycle that the texture coordinate read address is asserted.
RF0_read_addr: see above.
SP_TX_tc: Texture coordinate data sent from the shader pipe to the texture unit.
SEQ_TX_instr_start: Asserted on the first cycle of a SEQ to TX instruction transfer.
SEQ_TX_instruction: 96 bits of texture instruction transferred over 4 cycles.
SEQ_TX_clause: the clause number associated with this instruction.
SEQ_TX_write_addr: RF write index used by TX for returned texture data.
SEQ_TX_last: indicates that this is the last texture instruction of a clause.
SEQ_TX_phase: syncs the texture data write. Note that it is asserted early enough to be registered into TX and still allow TX to source the texture data to the SP on the correct cycle.

tx_phase: the phase signal after being registered into TX.
TX_SP_write_addr: RF write index for texture data.
TX_SP_valid: indicates that valid texture data is being driven to the SP.
TX_SP_data: the texture data.
TX_SEQ_clause: the clause number associated with the texture data.
TX_SEQ_done: indicates to the SEQ that the texture data transfer is complete for the clause number that is on the TX_SEQ_clause bus.

SEQ_SP_phase: see above under TD1 - shown here for reference.
SEQ_SP_write_addr: see above under TD1- shown here for reference.
RF0 write cycle: see above under TD1- shown here for reference.

~~8.~~15. Open issues

Formatted: Bullets and Numbering

There is currently an issue with constants. If the constants are not the same for the whole vector of vertices, we don't have the bandwith from the texture store to feed the ALUs. Two solutions exists for this problem:

1) Let the compiler handle the case and put those instructions in a texture clause so we can use the bandwith there to operate. This requires a significant amount of temporary storage in the register store.
2) Waterfall down the pipe allowing only at a given time the vertices having the same constants to operate in parrallel. This might in the worst case slow us down by a factor of 16.

Need to do some testing on the size of the ~~register file~~register file as well as on the ~~register file~~register file allocation method (dynamic VS static).

~~Ability to export at any clause?~~

Saving power?

~~Are we working on 32 vertices at a time or 16?~~

Size of the fifo containing the information of a vector of pixels/vertices. And size of the fifos before the reservation stations.

Sequencer Instruction memory, and constant memory.

Arbitration policy for the output file.

Loops and branches.

The parameter cache may end up in the PA rather than in the RS. Parameter cache management thus may change.

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001~~24~~ September, 200114	4 September, 2015~~3~~ October, 200124	GEN-CXXXXX-REVA	1 of 31

Author: Laurent Lefebvre

Issue To:	**Copy No:**

R400 Sequencer Specification

SEQ

Version 0.6~~5~~

Overview: This is an architectural specification for the R400 Sequencer block (SEQ). It provides an overview of the required capabilities and expected uses of the block. It also describes the block interfaces, internal sub-blocks, and provides internal state diagrams.

AUTOMATICALLY UPDATED FIELDS:

Document Location: C:\perforce\r400\arch\doc\gfx\RE\R400_Sequencer.doc

Current Intranet Search Title: R400 Sequencer Specification

APPROVALS	
Name/Dept	Signature/Date

Remarks:

THIS DOCUMENT CONTAINS CONFIDENTIAL INFORMATION THAT COULD BE SUBSTANTIALLY DETRIMENTAL TO THE INTEREST OF ATI TECHNOLOGIES INC. THROUGH UNAUTHORIZED USE OR DISCLOSURE.

ATI 2012
LG v. ATI
IPR2015-00326

Table Of Contents

1. OVERVIEW .... 4~~3~~
1.1 Top Level Block Diagram .... 6~~5~~
1.2 Data Flow graph .... 9~~7~~
1.3 Control Graph.................................. 11~~8~~
2. INTERPOLATED DATA BUS......... 11~~8~~
3. INSTRUCTION STORE .... 14~~11~~
4. SEQUENCER INSTRUCTIONS.... 14~~11~~
5. CONSTANT STORE..................... 14~~11~~
6. LOOPING AND BRANCHES........ 14~~11~~
6.1 Data dependant predicate instructions 16~~11~~
6.2 Register file indexing..................... 17~~12~~
7. REGISTER FILE ALLOCATION... 17~~12~~
8. TEXTURE ARBITRATION............ 18~~13~~
9. ALU ARBITRATION .... 18~~13~~
10. HANDLING STALLS .... 19~~14~~
11. CONTENT OF THE RESERVATION STATION FIFOS.................................. 19~~14~~
12. THE OUTPUT FILE (RB FIFO AND PARAMETER CACHE)......................... 19~~14~~
13. REGISTERS .... 19~~14~~
14. INTERFACES....................... 20~~14~~
14.1 External Interfaces............ 20~~14~~
14.1.1 Sequencer to Shader Engine Bus...................................... 20~~14~~
14.1.2 Shader Engine to Output File 20~~15~~
14.1.3 Shader Engine to Texture Unit Bus (Fast Bus) .... 20~~15~~
14.1.4 Sequencer to Texture Unit bus (Slow Bus) 22~~15~~
14.1.5 Shader Engine to RE/PA Bus 22~~15~~
14.1.6 PA? to sequencer.............. 22~~16~~
15. EXAMPLES OF PROGRAM EXECUTIONS...................................... 22~~16~~
15.1.1 Sequencer Control of a Vector of Vertices 22~~16~~
15.1.2 Sequencer Control of a Vector of Pixels 23~~17~~
15.1.3 Notes................................ 25~~18~~
16. OPEN ISSUES .... 31~~19~~
~~1. OVERVIEW... 3~~
~~1.1 Top Level Block Diagram .... 5~~
~~1.2 Data Flow graph .... 7~~
~~1.3 Control Graph...................................... 10~~
~~2. INTERPOLATED DATA BUS........... 10~~
~~3. INSTRUCTION STORE .... 13~~
~~4. CONSTANT STORE......................... 13~~
~~5. LOOPING AND BRANCHES............ 13~~
~~6. REGISTER FILE ALLOCATION....... 13~~
~~7. TEXTURE ARBITRATION................ 14~~
~~8. ALU ARBITRATION .... 14~~
~~9. HANDLING STALLS .... 15~~
~~10. CONTENT OF THE RESERVATION STATION FIFOS.. 15~~
~~11. THE OUTPUT FILE (RB FIFO AND PARAMETER CACHE)............................. 15~~
~~12. INTERFACES............................ 15~~
~~12.1 External Interfaces................. 15~~
~~12.1.1 Sequencer to Shader Engine Bus.. 15~~
~~12.1.2 Shader Engine to Output File 15~~
~~12.1.3 Shader Engine to Texture Unit Bus (Fast Bus) .... 16~~
~~12.1.4 Sequencer to Texture Unit bus (Slow Bus) 16~~
~~12.1.5 Shader Engine to RE/PA Bus 16~~
~~12.1.6 PA? to sequencer.................. 16~~
~~13. EXAMPLES OF PROGRAM EXECUTIONS.. 16~~
~~13.1.1 Sequencer Control of a Vector of Vertices 16~~
~~13.1.2 Sequencer Control of a Vector of Pixels 18~~
~~13.1.3 Notes...................................... 18~~
~~14. OPEN ISSUES.................................. 20~~
~~1. OVERVIEW... 3~~
~~1.1 Top Level Block Diagram .... 4~~
~~1.2 Data Flow graph .... 7~~
~~1.3 Control Graph...................................... 10~~

~~2. INTERPOLATED DATA BUS........... 10~~
~~3. INSTRUCTION STORE .... 10~~
~~4. CONSTANT STORE.......................... 11~~
~~5. LOOPING AND BRANCHES............ 11~~
~~6. REGISTER FILE ALLOCATION....... 11~~
~~7. TEXTURE ARBITRATION................. 12~~
~~8. ALU ARBITRATION .... 12~~
~~9. HANDLING STALLS .... 13~~
~~10. CONTENT OF THE RESERVATION STATION FIFOS.. 13~~
~~11. THE OUTPUT FILE (RB FIFO AND PARAMETER CACHE)............................. 13~~
~~12. INTERFACES............................. 13~~
~~12.1 External Interfaces................. 13~~
~~12.1.1 Sequencer to Shader Engine Bus.. 13~~
~~12.1.2 Shader Engine to Output File 13~~
~~12.1.3 Shader Engine to Texture Unit Bus (Fast Bus) .... 14~~
~~12.1.4 Sequencer to Texture Unit bus (Slow Bus) 14~~
~~12.1.5 Shader Engine to RE/PA Bus 14~~
~~12.1.6 PA? to sequencer.................. 14~~
~~13. EXAMPLES OF PROGRAM EXECUTIONS.. 14~~
~~13.1.1 Sequencer Control of a Vector of Vertices 14~~
~~13.1.2 Sequencer Control of a Vector of Pixels 16~~
~~13.1.3 Notes.................................... 17~~
~~14. OPEN ISSUES................................. 17~~

Revision Changes:

Rev 0.1 (Laurent Lefebvre) Date: May 7, 2001	First draft.
Rev 0.2 (Laurent Lefebvre) Date : July 9, 2001	Changed the interfaces to reflect the changes in the SP. Added some details in the arbitration section.
Rev 0.3 (Laurent Lefebvre) Date : August 6, 2001	Reviewed the Sequencer spec after the meeting on August 3, 2001.
Rev 0.4 (Laurent Lefebvre) Date : August 24, 2001	Added the dynamic allocation method for register file and an example (written in part by Vic) of the flow of pixels/vertices in the sequencer.
Rev 0.~~4~~ 5 (Laurent Lefebvre) Date : September 7, 2001	Added timing diagrams (Vic)
Rev 0.6 (Laurent Lefebvre) Date : September 24, 2001	Changed the spec to reflect the new R400 architecture.

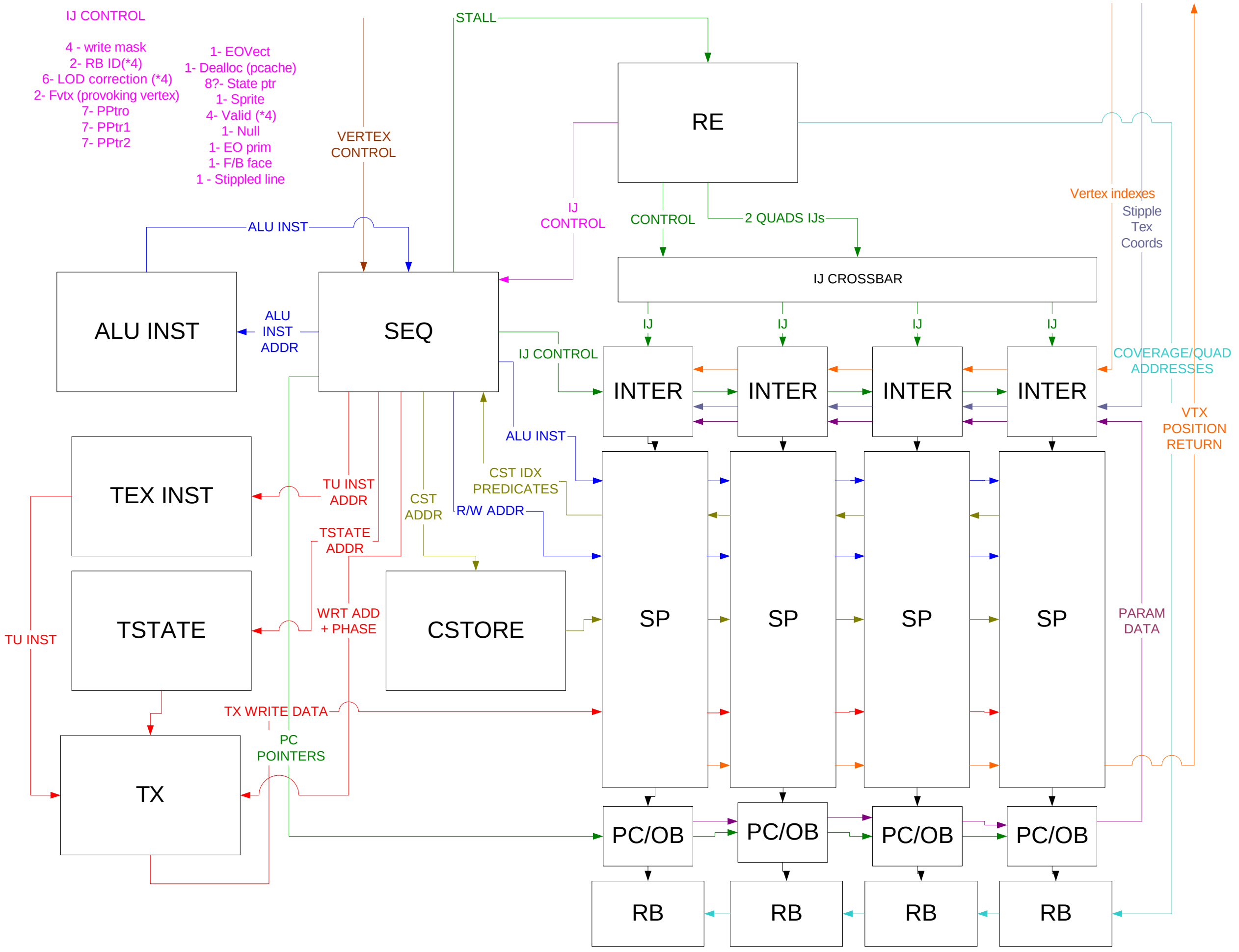
1. Overview

The sequencer first arbitrates between vectors of ~~16~~ 64 vertices that arrive directly from primitive assembly and vectors of ~~4~~ 16 quads (~~16~~ 64 pixels) that are generated in the raster engine.

The vertex or pixel program specifies how many GPR's it needs to execute. The sequencer will not start the next vector until the needed space is available.

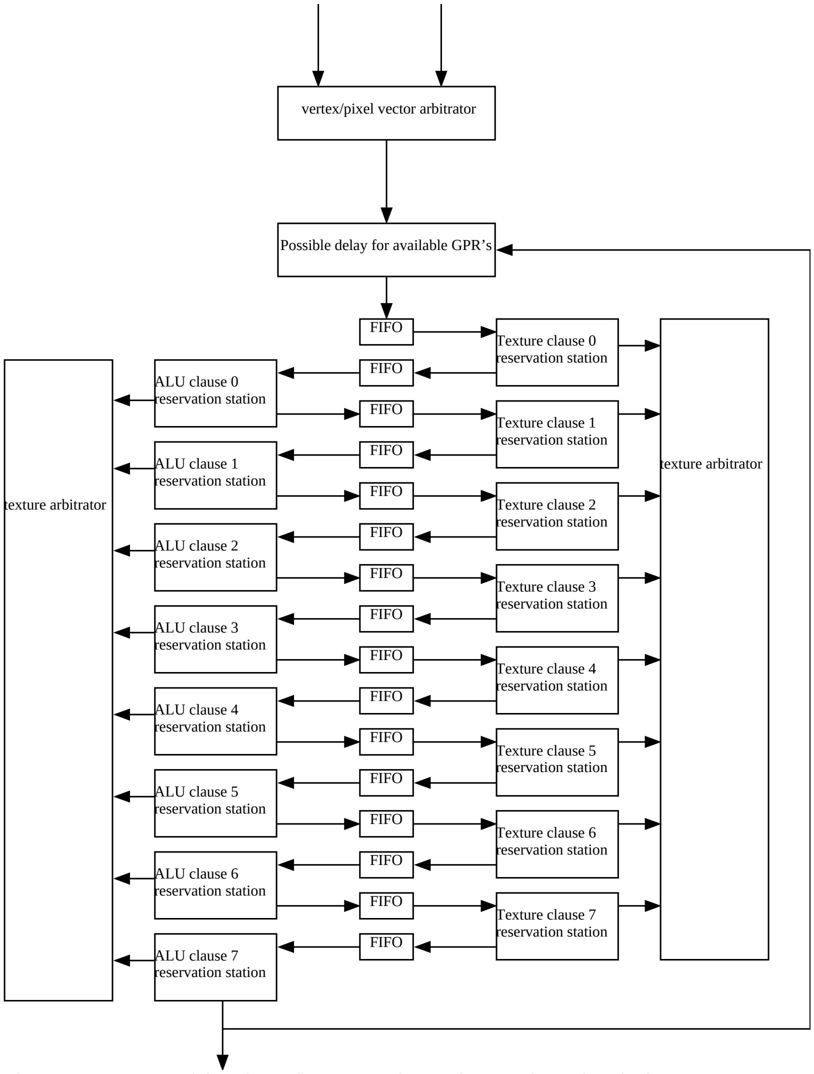
The sequencer is based on the R300 design. It chooses two ALU clauses and a texture clause to execute, and executes all of the instructions in a clause before looking for a new clause of the same type. Two ALU clauses are executed interleaved to hide the ALU latency. Each vector will have eight texture and eight ALU clauses, but clauses do not need to contain instructions. A vector of pixels or vertices ping-pongs along the sequencer FIFO, bouncing from texture reservation station to alu reservation station. A FIFO exists between each reservation stage, holding up vectors until the vector currently occupying a reservation station has left. A vector at a reservation station can be chosen to execute. The sequencer looks at all eight alu reservation stations to choose an alu clause to execute and all eight texture stations to choose a texture clause to execute. The arbitrator will give priority to clauses/reservation stations closer to the bottom of the pipeline. It will not execute an alu clause until the texture fetches initiated by the previous texture clause have completed. There are two separate sets of reservation stations, one for pixel vectors and one for vertices vectors. This way a pixel can pass a vertex and a vertex can pass a pixel.

To support the shader pipe the raster engine also contains the shader instruction cache and constant store. There are only one constant store for the whole chip and one instruction store. These will be shared among the four shader pipes. The four shader pipes also execute the same instuction thus there is only one sequencer for the whole chip.



IJ CONTROL
4 - write mask
2- RB ID(*4)
6- LOD correction (*4)
2- Fvtx (provoking vertex)
7- PPtro
7- PPtr1
7- PPtr2
1- EOVect
1- Dealloc (pcache)
8?- State ptr
1- Sprite
4- Valid (*4)
1- Null
1- EO prim
1- F/B face
1 - Stippled line
VERTEX CONTROL
STALL
RE
IJ CONTROL
CONTROL
2 QUADS IJs
Vertex indexes
Stipple Tex Coords
IJ CROSSBAR
ALU INST
ALU INST
SEQ
ALU INST ADDR
IJ
IJ
IJ
IJ
IJ CONTROL
INTER
INTER
INTER
INTER
COVERAGE/QUAD ADDRESSES
VTX POSITION RETURN
ALU INST
TEX INST
TU INST ADDR
CST ADDR
CST IDX PREDICATES
R/W ADDR
TSTATE ADDR
SP
SP
SP
SP
TSTATE
WRT ADD + PHASE
CSTORE
PARAM DATA
TU INST
TX WRITE DATA
PC POINTERS
TX
PC/OB
PC/OB
PC/OB
PC/OB
RB
RB
RB
RB

1.1 Top Level Block Diagram


vertex/pixel vector arbitrator
Possible delay for available GPR's
FIFO
texture arbitrator
ALU clause 0 reservation station
ALU clause 1 reservation station
ALU clause 2 reservation station
ALU clause 3 reservation station
ALU clause 4 reservation station
ALU clause 5 reservation station
ALU clause 6 reservation station
ALU clause 7 reservation station
Texture clause 0 reservation station
Texture clause 1 reservation station
Texture clause 2 reservation station
Texture clause 3 reservation station
Texture clause 4 reservation station
Texture clause 5 reservation station
Texture clause 6 reservation station
Texture clause 7 reservation station
texture arbitrator

There are two sets of the above figure, one for vertices and one for pixels.

The rasterizer always checks the vertices FIFO first and if allowed by the sequencer sends the data to the shader. If the vertex FIFO is empty then, the rasterizer takes the first entry of the pixel FIFO (a vector of ~~16~~ 64 pixels) and sends it to the interpolators. Then the sequencer takes control of the packet. The packet consists of ~~3~~ 20 bits of state, 6-7 bits for the base address of the Shader program and some information on the coverage to determine texture LOD. All other information (2x2 adresses) is put in a FIFO (one for the pixels and one for the vertices) and retrieved when the packet finishes its last clause.

{Issue: How many bits of state exactly?}

On receipt of a packet, the input state machine (not pictured but just before the first FIFO) allocated enough space in the registers to store the interpolated values and temporaries. Following this, the input state machine stacks the packet in the first FIFO.

On receipt of a command, the level 0 texture machine issues a texure request and corresponding register address for the texture address (ta). A small command (tcmd) is passed to the texture system identifying the current level number (0) as well as the register write address for the texture return data. One texture request is sent every 4 clocks causing the texturing of ~~four~~ sixteen 2x2s worth of data (or ~~16~~ 64 vertices). Once all the requests are sent the packet is put in FIFO 1.

Upon recept of the return data, the texture unit writes the data to the register file using the write address that was provided by the level 0 texture machine and sends the clause number (0) to the level 0 texture state machine to signify that the write is done and thus the data is ready. Then, the level 0 texture machine increments the counter of FIFO 1 to signify to the ALU 1 that the data is ready to be processed.

On receipt of a command, the level 0 ALU machine first decrements the input FIFO counter and then issues a complete set of level 0 shader instructions. For each instruction, the state machine generates 3 source addresses, one destination address (3 cycles later) and an instruction. Once the last instruction as been issued, the packet is put into FIFO 2.

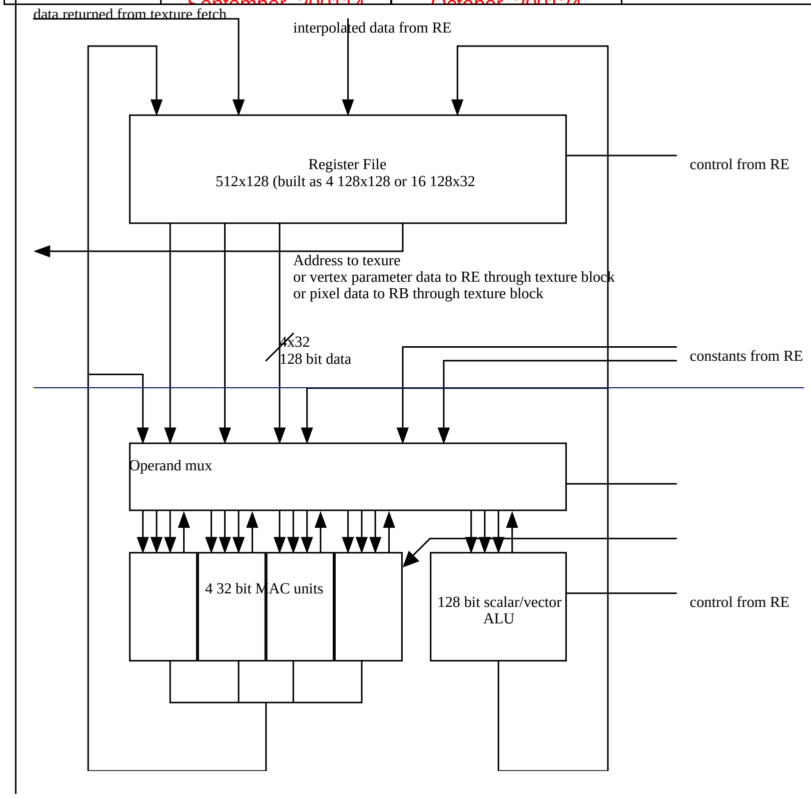
There will always be two active ALU clauses at any given time (and two arbitrers). One arbitrer will arbitrate over the odd clock cycles and the other one will arbitrate over the even clock cycles. The only constraints between the two arbitrers is that they are not allowed to pick the same clause number as they other one is currently working on if the packet os of the same type.

If the packet is a vertex packet, upon reaching ALU clause 4, it can export the position if the position is ready. So the arbitrer must prevent ALU clause 4 to be selected if the positional buffer is full (or can't be accessed). Along with the positional data, the location where the vertex data is to be put is also sent (parameter data pointers).

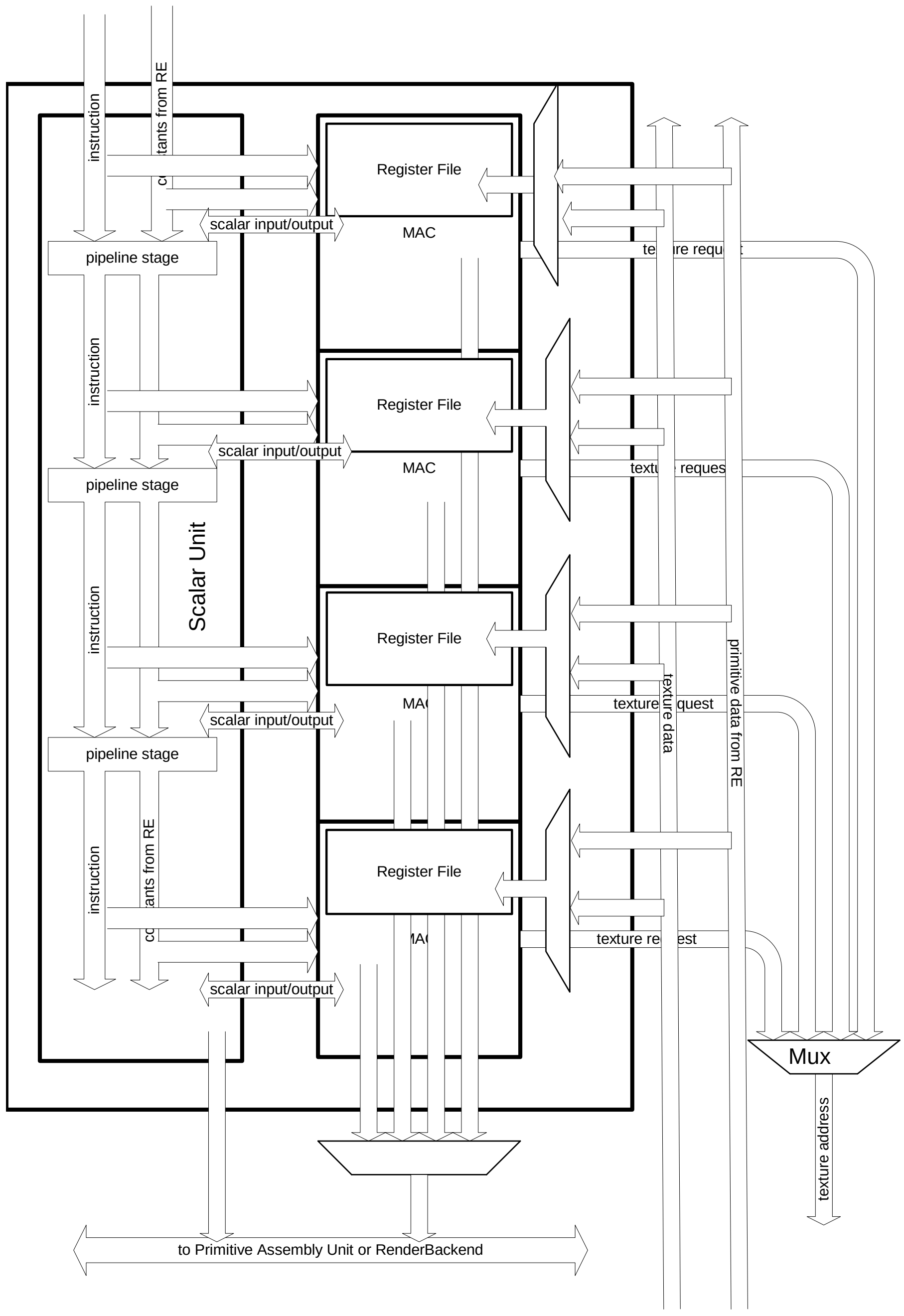
All other level process in the same way until the packet finally reaches the last ALU machine (8). On completion of the level 8 ALU clause, a valid bit is sent to the Render Backend which picks up the color data. This requires that the last instruction writes to the output register – a condition that is almost always true. If the packet was a vertex packet, instead of sending the valid bit to the RB, it is sent to the PA so it can know that the data present in the parameter store is valid.

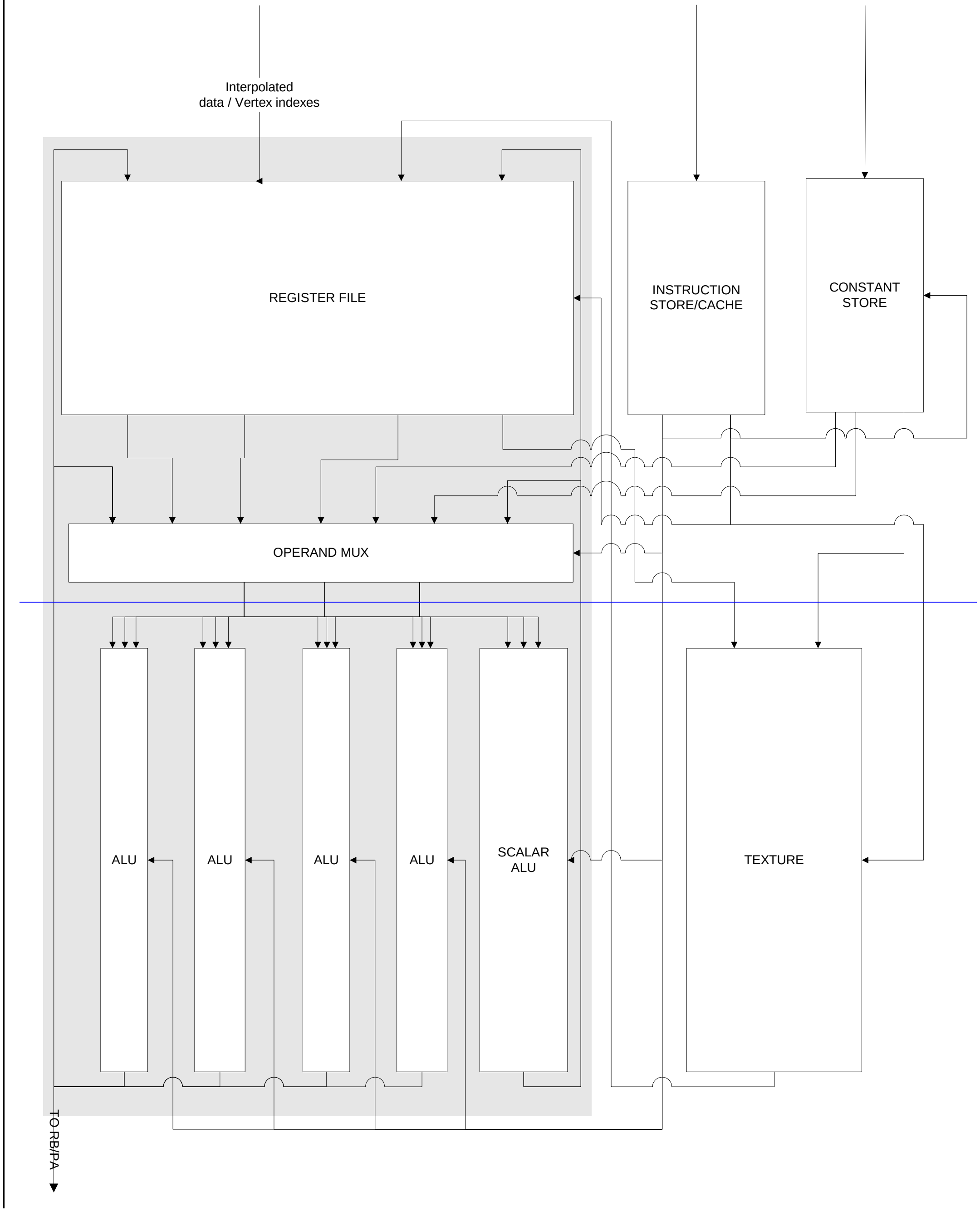
Only two ALU state machine may have access to the register file address bus or the instruction decode bus at one time. Similarly, only one texture state machine may have access to the register file address bus at one time. Arbitration is performed by three arbitrer blocks (two for the ALU state machines and one for the texture state machines). The arbitrers always favor the higher number state machines, preventing a bunch of half finished jobs from clogging up the register files.

~~Each state machine maintains an address pointer specifying where the 16 entries vector is located in the register file (the texture machine has two pointers one for the read address and one for the write). Upon completion of its job, the address pointer is incremented by a predefined amount equal to the total number of registers required by the shading code. A comparison of the address pointer for the first state machine in the chain (the input state machine), and the last machine in the chain (the level 8 ALU machine), gives an indication of how much unallocated register file memory is available~~


data returned from texture fetch
interpolated data from RE
Register File
512x128 (built as 4 128x128 or 16 128x32
control from RE
Address to texure
or vertex parameter data to RE through texture block
or pixel data to RB through texture block
4x32
128 bit data
constants from RE
Operand mux
4 32 bit MAC units
128 bit scalar/vector
ALU
control from RE

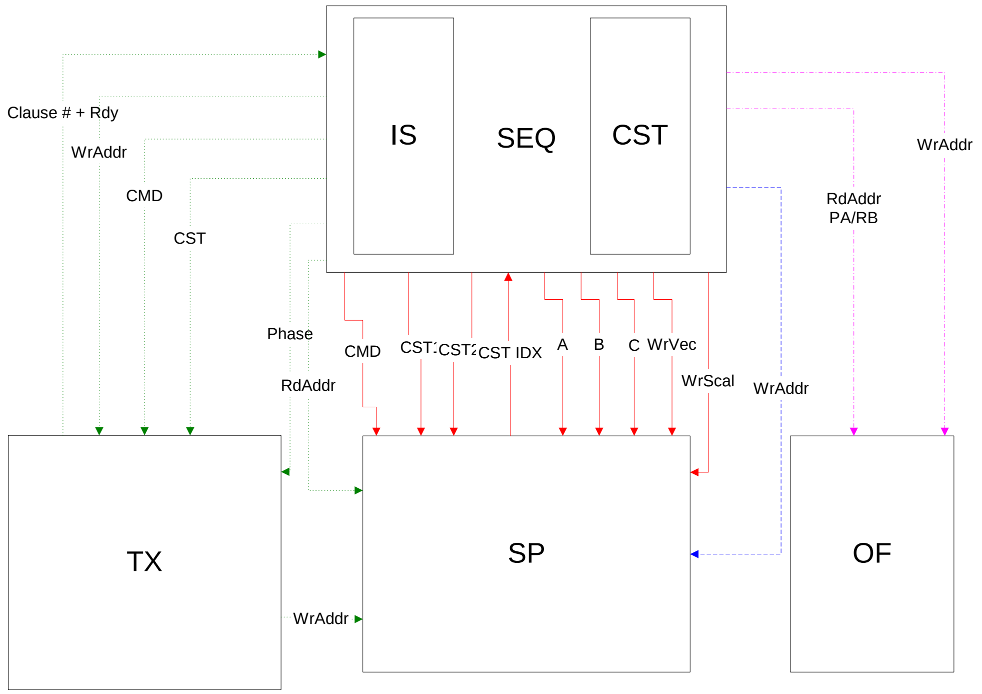
1.2 Data Flow graph





The gray area represents blocks that are replicated 4 times per shader pipe (16 times on the overall chip).

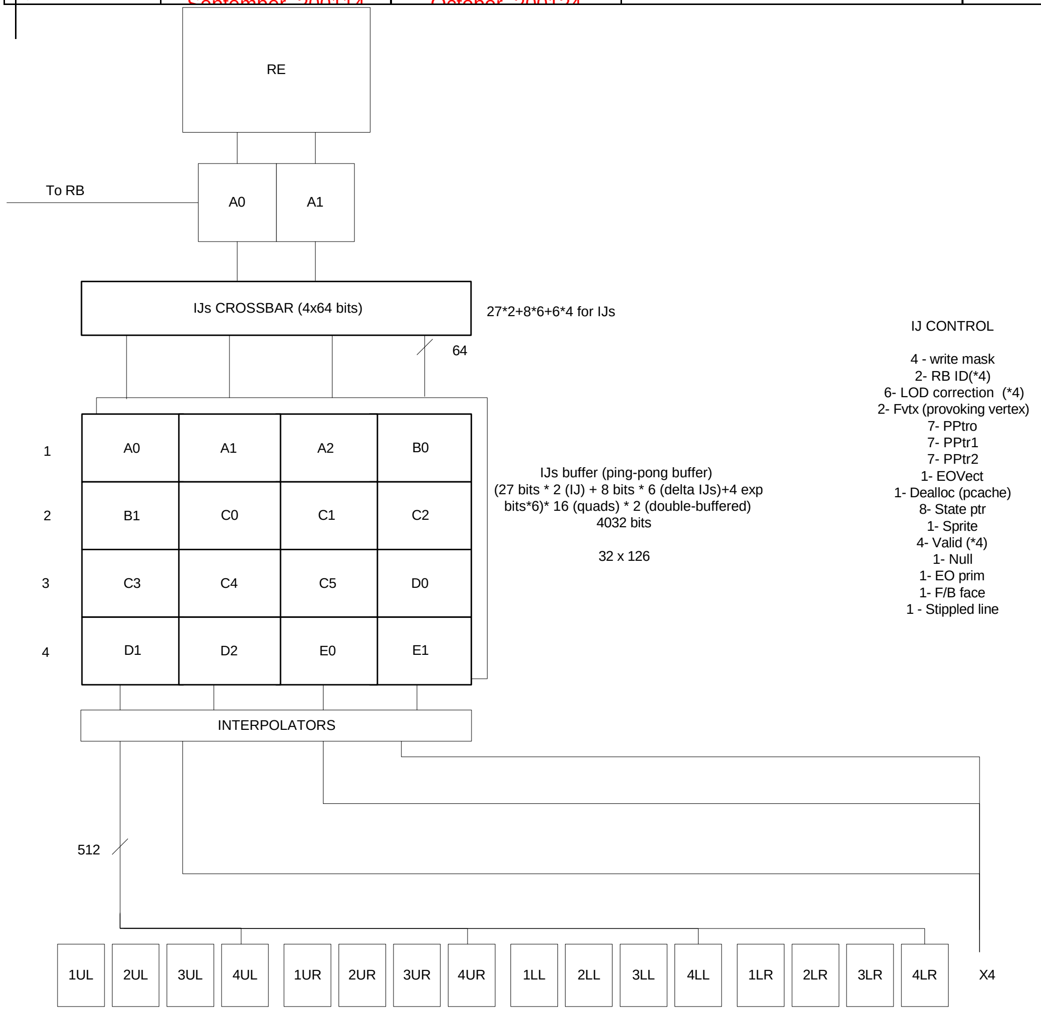
1.3 Control Graph



In green is represented the Texture control interface, in red the ALU control interface, in blue the Interpolated/Vector control interface and in purple is the output file control interface.

2. Interpolated data bus

The interpolators contain an IJ buffer to pack the information as much as possible before writing it to the register file.


RE
To RB
A0
A1
IJs CROSSBAR (4x64 bits)
27*2+8*6+6*4 for IJs
64
1
2
3
4
A0
A1
A2
B0
B1
C0
C1
C2
C3
C4
C5
D0
D1
D2
E0
E1
IJs buffer (ping-pong buffer)
(27 bits * 2 (IJ) + 8 bits * 6 (delta IJs)+4 exp
bits*6)* 16 (quads) * 2 (double-buffered)
4032 bits
32 x 126
IJ CONTROL
4 - write mask
2- RB ID(*4)
6- LOD correction (*4)
2- Fvtx (provoking vertex)
7- PPtro
7- PPtr1
7- PPtr2
1- EOVect
1- Dealloc (pcache)
8- State ptr
1- Sprite
4- Valid (*4)
1- Null
1- EO prim
1- F/B face
1 - Stippled line
INTERPOLATORS
512
1UL
2UL
3UL
4UL
1UR
2UR
3UR
4UR
1LL
2LL
3LL
4LL
1LR
2LR
3LR
4LR
X4

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19
SP0	A0	B1	C3	D1					A0	B1	C3	D1					V 0-3	V 16-19	V 32-35	V 48-51
SP1	A1		C4	D2		C0			A1		C4	D2		C0			V 4-7	V 20-23	V 36-39	V 52-55
SP2	A2		C5			C1		E0	A2		C5			C1		E0	V 8-11	V 24-27	V 40-43	V 56-59
SP3					B0	C2	D0	E1					B0	C2	D0	E1	V 12-15	V 28-31	V 44-47	V 60-63

P0

P1

Above is an example of a tile we might receive. The IJ information is packed in the IJ buffer 2 quads at a time. The sequencer allows at any given time as many as four quads to interpolate a parameter. They all have to come from the same primitive. Then the sequencer controls the write mask to the register to write the valid data in.~~Since each of the register file is actually physically divided (one 128x128 per MAC) and we don't have the place to hold a maximum size vector of vertices in the parameter buffer, we need to interpolate on a parameter basis rather than on a quad basis. So the order to the register file will be:~~

~~Q0P0 Q1P0 Q2P0 Q3P0 Q0P1 Q1P1 Q2P1 Q3P1 Q0P2 Q1P2 …~~

3. Instruction Store

There is going to be only one instruction store for the whole chip. It ~~may~~ will contain ~~up to 200~~40960 instructions of 96 bits each. There is also going to be a control instruction store of 256x32.

{ISSUE : The instruction store is loaded by the sequencer using the memory hub ?}.

The read bandwith from this store is ~~24~~ 96*2 bits/ 4 clocks (48 bits/clock)~~/pipe~~. ~~To achieve this this instruction store is likely to be broken up into 4 blocks. An ALU instruction section (1R/1W) split in two and a texture section (1R/1W) also split in two.~~ The bandwith out of those memories is ~~96~~ 48 bits/clock. It is likely to be a 1R/1W port memory; we use 2 clocks to load the ALU instruction and 2 clocks to load the Texture instruction.

Formatted: Bullets and Numbering

4. Sequencer Instructions

All control flow instructions and move instructions are handled by the sequencer only. The ALUs will perform NOPs during this time (MOV PV,PV, PS,PS).

Formatted: Bullets and Numbering

~~4.~~5. Constant Store

The constant store is managed by the CP. The sequencer is aware of where the constants are using a remaping table also managed by the CP. A likely size for the constant store is 512x128 bits. The constant store is also planned to be shared. The read BW from the constant store is 128~~512/4~~ bits/clock~~/pipe~~ and the write bandwith is 32/4 bits/clock.

In order to do constant store indexing, the sequencer must be loaded first with the indexes (that come from the GPRs). There are 144 wires from the exit of the SP to the sequencer (9 bits pointers x 16 vertexes/clock). Since the data must pass thru the Shader pipe for the float to fixed convertion, there is a latency of 4 clocks (1 instruction) between the time the sequencer is loaded and the time one can index into the constant store. The assembly will look like this

```
MOVA  R1.X,R2.X        // Loads the sequencer with the content of R2.X, also copies the content of R2.X into R1.X
NOP                    // latency of the float to fixed conversion
ADD   R3,R4,C0[R2.X]// Uses the state from the sequencer to add R4 to C0[R2.X] into R3
```

Note that we don't really care about what is in the brackets because we use the state from the MOVA instruction. R2.X is just written again for the sake of simplicity.

The storage needed in the sequencer in order to support this feature is 2*64*9 bits = 1152 bits.

Formatted: Bullets and Numbering

~~5.~~6. Looping and Branches

Loops and branches are planned to be supported and will have to be dealt with at the sequencer level. We plan on supporting constant loops and branches using a control program. The control program has 4 instructions:

Formatted: Bullets and Numbering

6.1 The controlling state.

As per Dx9 the following state is available for control flow:

Boolean[15:0]

loop_count[7:0][7:0]
In addition:
loop_start [7:0] [7:0]
loop_step [7:0] [7:0]
Exist to give more control to the controlling program.

We will extend that in the R400 to:
Boolean[31:0]
Loop_count[7:0][15:0]
Loop_Start[7:0] [15:0]
Loop_End[7:0] [15:0]

6.2 The Control Flow Program

Formatted: Bullets and Numbering

The R300 uses a match method for control flow: The shader is executed, and at every instruction its address is compared with addresses (or address?) in a control table. The "event" in the control table can redirect operations in the program. I believe that this method has increased area and complexity when the program size is increased.

The Method I prefer is a "control program"
The control program has four basic instructions:
Execute
Conditional_execute
Loop_start
Loop_end

Execute, causes the specified number of instructions in instruction store to be executed.
Conditional_execute checks a condition first, and if true, causes the specified number of instructions in instruction store to be executed.
Loop_start resets the corresponding loop counter to the start value.
Loop_end increments (decrements?) the loop counter and jumps back the specified number of instructions if the loop end condition is not met.

if we try and fit the control flow instructions into 32 bit words, the following instructions are possible choices:

Execute																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	instruction_count												Reserved			Address												

Execute up to 4K instructions at the specified address in the instruction memory.

Conditional Execute																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	Boolean						0: = 0 1: = 1 2,3:NA		Instruction_count							Address												

if the specified boolean (6 bits can address 64 booleans) meets the specified condition then execute the specified instructions (up to 64 instructions)

Conditional Execute Predicates																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	Reserved =0						0: = 0 1: = 1 2,3:NA		Instruction_count							Address												

Check the OR of all current predicate bits. If OR matches the condition execute the specified number of instructions.

Loop_Start																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

0	0	1	0																								Loop ID				

Initialize the specified loop

Loop_End																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	Reserved (must = 0)												Start Address								Reserved (must = 0)			Loop ID				

if the loop condition of the current loop is not met, then branch back to the specified address in the control flow program. Note that jumping back to the loop_start results in an infinite loop, the jump should be to loop_start+1.

the way this is described does not prevent nested loops, and the inclusion of the loop id make this easy to do.

The basic model is as follows:

The render state defined the clause boundaries:
Vertex_shader_fetch[7:0][7:0]   // eight 8 bit pointers to the location where each clauses control program is located
Vertex_shader_alu[7:0][7:0]   // eight 8 bit pointers to the location where each clauses control program is located
Pixel_shader_fetch[7:0][7:0]   // eight 8 bit pointers to the location where each clauses control program is located
Pixel_shader_alu[7:0][7:0]   // eight 8 bit pointers to the location where each clauses control program is located

The control program can be up to 256 instructions in size. (There is an offset added to the address from the render state before accessing the control flow program memory to allow for multiple programs resident at the same time)

The control program for a given clause is executed to completion before moving to another clause, (with the exception of the pick two nature of the alu execution)

The addresses from the control program are added to another offset to allow for multiple programs resident at the same time.

Under this model, all subroutine calls must be inlined into the control program.

6.3 Data dependant predicate instructions

Formatted: Bullets and Numbering

Data dependant conditionals will be supported in the R400. The only way we plan to support those is by supporting three vector/scalar predicate operations of the form:

PRED_SETE - similar to SETE except that the result is 'exported' to the sequencer.
PRED_SETGT - similar to SETGT except that the result is 'exported' to the sequencer
PRED_SETGTE - similar to SETGTE except that the result is 'exported' to the sequencer

For the scalar operations only we will also support the two following instructions:
PRED_SETE0 – SETE0
PRED_SETE1 – SETE1

The export is a single bit - 1 or 0 that is sent using the same data path as the MOVA instruction. The sequencer will maintain the 64 bit predicate vector and use it to control the write masking (two sets for interleaved operation). This predicate is not maintained across clause boundaries.

Then we have two conditional execute bits. The first bit is a conditional execute “on” bit and the second bit tells us if we execute on 1 or 0. For exemple, the instruction :

P0_ADD R0,R1,R2

Is only going to write the result of the ADD into those GPRs whose predicate bit is 0. Alternatively, P1_ADD would only write the results to the GPRs whose predicate bit is set. The use of the P0 or P1 without precharging the sequencer with a PRED instruction is undefined.

{Issue: do we have to have a NOP between PRED and the first instruction that uses a predicate?}

6.4 Register file indexing

Formatted: Bullets and Numbering

Because we can have loops in texture clause, we need to be able to index into the register file in order to retrieve the data created in a texture clause loop and use it into an ALU clause. The instruction will include the base address for register indexing and the instruction will contain these controls :

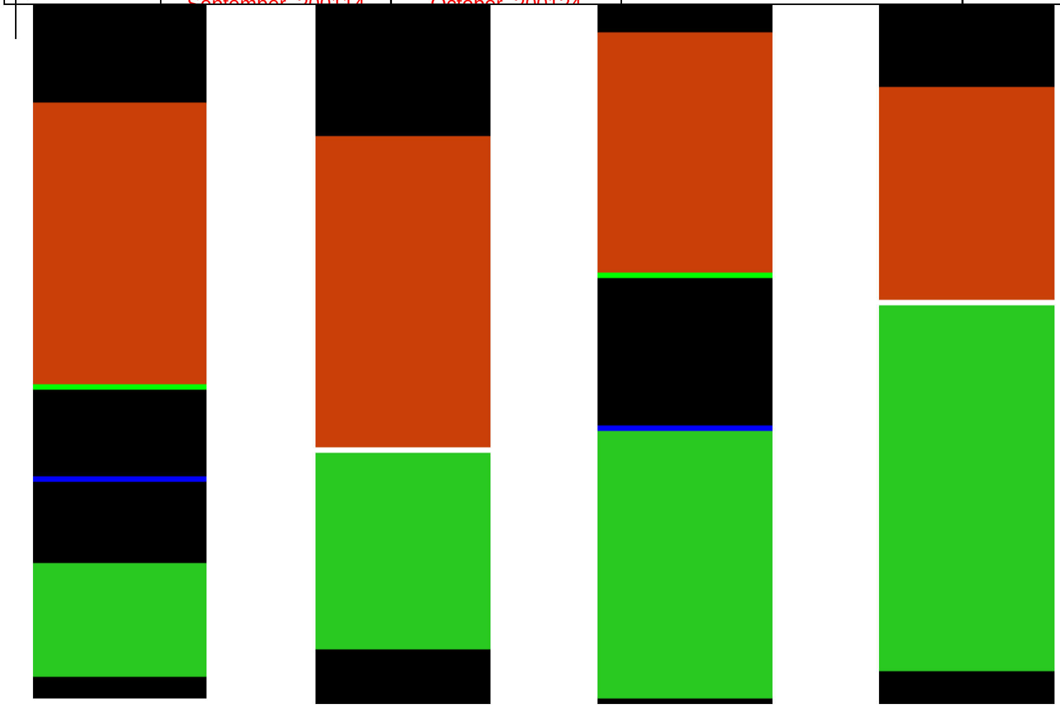
Bit7	Bit 6	
0	0	'absolute register'
0	1	'relative register'
1	0	'previous vector'
1	1	'previous scalar'

In the case of an absolute register we just take the address as is. In the case of a relative register read we take the base address and we add to it the loop_counter and this becomes our new address that we give to the shader pipe.~~However, it is still unclear if we plan on supporting data dependent branches or not.~~

~~6.~~7. Register file allocation

Formatted: Bullets and Numbering

The register file allocation for vertices and pixels can either be static or dynamic. In both cases, the register file in managed using two round robins (one for pixels and one for vertices). In the dynamic case the boundary between pixels and vertices is allowed to move, in the static case it is fixed to VERTEX_REG_SIZE for vertices and 256-VERTEX_REG_SIZE for pixels.



Above is an example of how the algorithm works. Vertices come in from top to bottom; pixels come in from bottom to top. Vertices are in orange and pixels in green. The blue line is the tail of the vertices and the green line is the tail of the pixels. Thus anything between the two lines is shared. When pixels meets vertices the line turns white and the boundary is static until both vertices and pixels share the same "unallocated bubble". Then the boundary ~~as~~ is allowed to moving~~e~~ again.

Formatted: Bullets and Numbering

~~7.~~8. Texture Arbitration

The texture arbitration logic chooses one of the 8 potentially pending texture clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute. Once chosen, the clause state machine will send one 2x2 texture fetch per clock (or 4 fetches in one clock every 4 clocks) until all the texture fetch instructions of the clause are sent. This means that there cannot be any dependencies between two texture fetches of the same clause.

The arbitrator will not wait for the texture fetches to return prior to selecting another clause for execution. The texture pipe will be able to handle up to X(?) in flight texture fetches and thus there can be a fair number of active clauses waiting for their texture return data.

Formatted: Bullets and Numbering

~~8.~~9. ALU Arbitration

ALU arbitration proceeds in almost the same way than texture arbitration. The ALU arbitration logic chooses one of the 8 potentially pending ALU clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute. There are two ALU arbitrers, one for the even clocks and one for the odd clocks. For exemple, here is the sequencing of two interleaved ALU clauses (E and O stands for Even and Odd):

Einst0 Oinst0 Einst1 Oinst1 Einst2 Oinst2 Einst0 Oinst3 Einst1 Oinst4 Einst2 Oinst0…
Proceeding this way hides the latency of 8 clocks of the ALUs.

~~9.~~10. Handling Stalls

Formatted: Bullets and Numbering

When the output file is full, the sequencer prevents the ALU arbitration logic to select the last clause (this way nothing can exit the shader pipe until there is place in the output file. If the packet is a vertex packet and the position buffer is full (POS_FULL) then the sequencer also prevents a thread to enter the exporting clause (4?). The sequencer will set the OUT_FILE_FULL signal n clocks before the output file is actually full and thus the ALU arbitrer will be able read this signal and act accordingly by not preventing exporting clauses to proceed.

~~10.~~11. Content of the reservation station FIFOs

Formatted: Bullets and Numbering

3 bits of Render State 6-7 bits for the base address of the instruction store, ~~and~~ some bits for LOD correction and coverage mask information in order to fetch texture for only valid pixels. Every other information (such as the coverage mask, quad address, etc.) is put in a FIFO and is retrieved when the quad exits the shader pipe to enter in the output file buffer. Since pixels and vertices are kept in order in the shader pipe, we only need two fifos (one for vertices and one for pixels) deep enough to cover the shader pipe latency. This size will be determined later when we will know the size of the small fifos between the reservation stations.

~~11.~~12. The Output File (RB FIFO and Parameter Cache)

Formatted: Bullets and Numbering

The output file is where program results are exported when the pixel/vertex shader finishes. It constists of a 512x128 memory cell that is statically divided between pixels and vertices. The output file has 1 write port and 1 read port. The sequencer is responsible for managing the addresses of this output file and for stalling the shader pipe should this output file fill up. The management is done by keeping the tail and head pointers of each sections (pixels and vertices) and incrementing them using a simple RoundRobin allocation policy. The sequencer must also arbitrate between the PA and the RB for the use of the read port. This arbitration will either be priority based or just interleaved evenly (1 read every 2 clocks for each of the blocks).

13. Registers

Formatted: Bullets and Numbering

DYNAMIC_REG	Dynamic allocation (pixel/vertex) of the register file on or off.
VERTEX_REG_SIZE	What portion of the register file is reserved for vertices (static allocation only)
PIXEL_MIN_SIZE	Minimal size of the register file's pixel portion (dynamic only)
VERTEX_MIN_SIZE	Minimal size of the register file's vertex portion (dynamic only)
Vshader_fetch[7:0][7:0]	eight 8 bit pointers to the location where each clauses control program is located
Vshader_alu[7:0][7:0]	eight 8 bit pointers to the location where each clauses control program is located
Pshader_fetch[7:0][7:0]	eight 8 bit pointers to the location where each clauses control program is located
Pshader_alu[7:0][7:0]	eight 8 bit pointers to the location where each clauses control program is located
PSHADER	base pointer for the pixel shader
VSHADER	base pointer for the vertex shader
PCNTLSHADER	base pointer for the pixel control program
VCNTLSHADER	base pointer for the vertex control program
VWRAP	wrap point for the vertex shader instruction store
PWRAP	wrap point for the pixel shader instruction store
REG_ALLOC_PIX	number of registers to allocate for pixel shader programs
REG_ALLOC_VERT	number of registers to allocate for vertex shader programs
PARAM_MASK[0…16]	parameter mask to specify wich parameters the pixel shader
FLAT_GOUR[0…16]	wich parameters are to be gouraud shaded
GEN_TEX[0….16]	for wich parameters do we need to generate tex coords.
CYL_WRAP[0…64]	for wich vertices do we do the cyl wrapping.
P_EXPORT	number of exports for pixel shader
V_EXPORT	number of exports for vertex shader (also the number of interpolated parameters)
V_EXPORT_LOC	Vertex shader exporting to RB or the PCACHE

~~12.~~14. Interfaces

Formatted: Bullets and Numbering

~~12.1~~14.1 External Interfaces

~~12.1.1~~14.1.1 *~~Sequencer to Shader Engine Bus~~PA/SC to RE : IJ bus*

~~This is a bus that sends the instruction and constant data to all 4 Sub-Engines of the Shader. Because a new instruction is needed only every 4 clocks, the width of the bus is divided by 4 and both constants and instruction are sent over those 4 clocks.~~ This is a bus that sends the IJ information to the IJ fifos on the top of each shader pipe. At the same time the control information goes to the sequencer

Name	Direction	Bits	Description
~~Instruction Start~~IJs	~~SEQ-> SP~~PA→RE	~~1~~64	~~High on first cycle of transfer~~IJ information sent over 2 clocks

Formatted: Bullets and Numbering

~~12.1.2~~

~~*12.1.2Shader Engine to Output File*~~

~~Every clock each Sub-Engine can output 128 bits of 'vector' data and 32 bits of 'scalar' data to an output file (?). This data will be compressed into 128 bits total prior to storage in output file.~~

14.1.2 *PA/SC to SEQ : IJ Control bus*

Formatted: Bullets and Numbering

This is the control information sent to the sequencer in order to control the IJ fifos and all other information needed to execute a shader program on the sent pixels.

Name	**Direction**	**Bits**	**Description**
Write Mask	PA→SEQ(RE)	4	Quad Write mask left to right
RB_ID	PA→SEQ(RE)	8	RB id for each quad sent 2 bits per quad
LOD_CORRECT	PA→SEQ(RE)	24	LOD correction per quad (6 bits per quad)
FVTX	PA→SEQ(RE)	2	Provoking vertex for flat shading
PPTR0	PA→SEQ(RE)	11	P Store pointer for vertex 0
PPRT1	PA→SEQ(RE)	11	P Store pointer for vertex 1
PPTR2	PA→SEQ(RE)	11	P Store pointer for vertex 2
E_OFF_VECTOR	PA→SEQ(RE)	1	End of the vector
DEALLOC	PA→SEQ(RE)	1	Deallocation token for the P Store
STATE	PA→SEQ(RE)	21	State/constant pointer (6*3+3)
SPRITE	PA→SEQ(RE)	1	Need to generate tex cords
VALID	PA→SEQ(RE)	16	Valid bits for all pixels
NULL	PA→SEQ(RE)	1	Null Primitive (for deallocation purposes)
E_OFF_PRIM	PA→SEQ(RE)	1	End Of the primitive
FBFACE	PA→SEQ(RE)	1	Front face = 1, back face = 0
STIPPLE_LINE	PA→SEQ(RE)	1	Stippled line need to load tex cords from alternate buffer
RTRn	SEQ→PA	1	Stalls the PA in n clocks
RTS	PA→SEQ(RE)	1	PA ready to send data

14.1.3 *PA/SC to RE : Vertex Bus*

Formatted: Bullets and Numbering

Name	**Direction**	**Bits**	**Description**

14.1.4 PA/SC to SEQ : Vertex Control Bus

Formatted: Bullets and Numbering

Name	**Direction**	**Bits**	**Description**
STATE	PA→SEQ	21	Render State (6*3+3 for constants)

14.1.5 CP to SEQ : Constant store load

Formatted: Bullets and Numbering

Name	**Direction**	**Bits**	**Description**
Constant Address	CP→SEQ	8	Address of the block of 4 constants
Constant Data	CP→SEQ	512	Data sent over X clocks
Remap Address	CP→SEQ	10	Remaping address write address
Remap Data pointer	CP→SEQ	8	Remaping pointer

14.1.6 CP to SEQ : Texture State store load

Formatted: Bullets and Numbering

Name	**Direction**	**Bits**	**Description**
Constant Address	CP→SEQ	8	Address of the block of 4 state constants
Constant Data	CP→SEQ	512	Data sent over X clocks
Remap Address	CP→SEQ	10	Remaping address write address
Remap Data pointer	CP→SEQ	8	Remaping pointer

14.1.7 CP to SEQ : Control State store load

Formatted: Bullets and Numbering

Name	**Direction**	**Bits**	**Description**

14.1.8 MH to SEQ: Instruction store Load

Formatted: Bullets and Numbering

Name	**Direction**	**Bits**	**Description**
Instruction address	MH→SEQ	12	Instruction address
Instruction	MH→SEQ	96	Instruction X times
Control Instruction address	MH→SEQ	8	Pointer to the instruction store
Control Instruction	MH→SEQ	32	Control Instruction X times

14.1.9 OB to RB : Pixel read from RBs

Formatted: Bullets and Numbering

Name	**Direction**	**Bits**	**Description**
Pixel Data	OB→RB	128	2 pixels (or ½ quad)
Quad Address	OB→RB	20	XY address 10 bits per

14.1.10 SP to PA/SC : Position return bus

Formatted: Bullets and Numbering

Name	**Direction**	**Bits**	**Description**
Position return	SP→PA	128	Position data or sprite size
Position Buffer pointer	SP→PA	7?	Pointer to the position cache
Parameter cache pointer	SP→PA	11	Pointer where the data will be in the parameter cache

~~12.1.3~~14.1.11 Shader Engine to Texture Unit Bus (Fast Bus)

Formatted: Bullets and Numbering

~~One~~ Four quad's worth of addresses is transferred to Texture Unit every clock. These are sourced from a different pixel within each of the sub-engines repeating every 4 clocks. The register file index to read must precede the data by 2 clocks. The Read address associated with Quad 0 must be sent 1 clock after the Instruction Start signal is sent, so that data is read 3 clocks after the Instruction Start.

~~One~~Four Quad's worth of Texture Data may be written to the Register file every clock. These are directed to a different pixel of the sub-engines repeating every 4 clocks. The register file index to write must accompany the data. Data and Index associated with the Quad 0 must be sent 3 clocks after the Instruction Start signal is sent.

Name	Direction	Bits	Description
Tex_Read_Register_Index	SEQ->SP	8	Index into Register files for reading Texture Address
Tex_RegFile_Read_Data	SP->TEX	~~512~~2048	~~4~~ 16 Texture Addresses read from the Register file
Tex_Write_Register_Index	SEQ->TEX	8	Index into Register file for write of returned Texture Data

~~12.1.4~~14.1.12 *Sequencer to Texture Unit bus (Slow Bus)*

Formatted: Bullets and Numbering

Once every four clock, the texture unit sends to the sequencer on wich clause it is now working and if the data in the registers is ready or not. This way the sequencer can update the texture counters for the reservation station fifos. The sequencer also provides the intruction and constants for the texture fetch to execute and the address in the register file where to write the texture return data.

Name	Direction	Bits	Description
Tex_Ready	TEX→ SEQ	1	Data ready
Tex_Clause_Num	TEX→ SEQ	3	Clause number
Tex_cst	SEQ→TEX	~~?~~10	Texture ~~constants X~~state address 10 bits sent over 4 clocks
Tex_Inst	SEQ→TEX	~~?~~12	Texture fetch instruction ~~X~~address 12 bits sent over 4 clocks

~~12.1.5 *Shader Engine to RE/PA Bus*~~

Formatted: Bullets and Numbering

~~12.1.6 *PA? to sequencer*~~

~~13.~~15. Examples of program executions

~~13.1.1~~15.1.1 *Sequencer Control of a Vector of Vertices*

1. PA sends a vector of ~~16~~ 64 vertices (actually vertex indices – 32 bits/index for ~~512~~ 2048 bit total) to the RE's Vertex FIFO
   - state pointer as well as tag into position cache is sent along with vertices
   - space was allocated in the position cache for transformed position before the vector was sent
   - **also before the vector is sent to the RE, the CP has loaded the global instruction store with the vertex shader program (using the MH?)**
   - The vertex program is assumed to be loaded when we receive the vertex vector.
     - the SEQ then accesses the IS base for this shader using the local state pointer (provided to all sequencers by the RBBM when the CP is done loading the program)
2. SEQ arbitrates between the Pixel FIFO and the Vertex FIFO – basically the Vertex FIFO always has priority
   - at this point the vector is removed from the Vertex FIFO
   - the arbitrer is not going to select a vector to be transformed if the parameter cache is full unless the pipe as nothing else to do (ie no pixels are in the pixel fifo).
3. SEQ allocates space in the SP register file for index data plus GPRs used by the program
   - the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer that came down with the vertices
   - SEQ will not send vertex data until space in the register file has been allocated
4. SEQ sends the vector to the SP register file over the RE_SP interface (which has a bandwidth of ~~512~~ 2048 bits/cycle)
   - the ~~16~~ 64 vertex indices are sent to the ~~16~~ 64 register files over 4 cycles
     - RF0 of SU0, SU1, SU2, and SU3 is written the first cycle
     - RF1 of SU0, SU1, SU2, and SU3 is written the second cycle
     - RF2 of SU0, SU1, SU2, and SU3 is written the third cycle

    - RF3 of SU0, SU1, SU2, and SU3 is written the fourth cycle
  - the index is written to the least significant 32 bits **(floating point format?) (what about compound indices)** of the 128-bit location within the register file (w); the remaining data bits are set to zero (x, y, z)

5. SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of texture state machine 0, or TSM0 FIFO)
   - the control packet contains the state pointer, the tag to the position cache and a register file base pointer.
6. TSM0 accepts the control packet and fetches the instructions for texture clause 0 from the global instruction store
   - TSM0 was first selected by the TSM arbiter before it could start
7. all instructions of texture clause 0 are issued by TSM0
8. the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)
   - TSM0 does not wait for requests made to the Texture Unit to complete; it passes the register file write index for the texture data to the TU, which will write the data to the RF as it is received
   - once the TU has written all the data to the register files, it increments a counter that is associated with ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go ahead start to execute the ALU clause
9. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store
10. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of texture state machine 1, or TSM1 FIFO)
11. the control packet continues to travel down the path of reservation stations until all clauses have been executed
    - position can be exported in ALU clause 3 (or 4?); the data (and the tag) is sent over a position bus (which is shared with all four shader pipes) back to the PA's position cache
    - A parameter cache pointer is also sent along with the position data. This tells to the PA where the data is going to be in the parameter cache.
      - there is a position export FIFO in the SP that buffers position data before it gets sent back to the PA
      - the ASM arbiter will prevent a packet from starting an exporting clause if the position export FIFO is full
    - parameter data is exported in clause 7 (as well as position data if it was not exported earlier)
      - parameter data is sent to the Parameter Cache over a dedicated bus
      - the SEQ allocates storage in the Parameter Cache, and the SEQ deallocates that space when there is no longer a need for the parameters (it is told by the PA when using a token).
      - the ASM arbiter will prevent a packet from starting on ASM7 if the parameter cache (or the position buffer if position is being exported) is full
12. after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program

~~13.1.2~~15.1.2 *Sequencer Control of a Vector of Pixels*

Formatted: Bullets and Numbering

1. **As with vertex shader programs, pixel shaders are loaded into the global instruction store by the CP**
   - At this point it is assumed that the pixel program is loaded into the instruction store and thus ready to be read.

~~2.the RE's Parameter Buffer is loaded from the Parameter Cache before the SEQ takes control of the vector~~

- ~~after the HZ culling stage a request is made by the RE to send parameter data to the Parameter buffer~~
- ~~the Parameter buffer is wide enough to source 3 vertices worth of a particular parameter in one cycle~~
- ~~**at this moment the right sequencer will free up the parameter store locations not used anymore using the token provided by the PA**.~~

Formatted: Bullets and Numbering

~~3.~~2. the RE's Pixel FIFO is loaded with the barycentric coordinates for pixel quads by the detailed walker
   - the state pointer and the LOD correction bits are also placed in the Pixel FIF0
   - the Pixel FIFO is wide enough to source ~~one~~ four quad's worth of barycentrics per cycle

Formatted: Bullets and Numbering

~~4.~~3. SEQ arbitrates between Pixel FIFO and Vertex FIFO – when there are no vertices pending OR there is no space left in the register files for vertices, the Pixel FIFO is selected

~~5.~~4. SEQ allocates space in the SP register file for all the GPRs used by the program

- the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer
- SEQ will not allow interpolated data to be sent to the shader until space in the register file has been allocated

~~6.SEQ control starts with the interpolation of parameters (up to 16 per thread) by sending the barycentric coordinates from the Pixel FIFO and the parameters from the Parameter Buffer to the interpolator~~

Formatted: Bullets and Numbering

- ~~P0i, P0j, and P0k (the value of P0 at each vertex) are loaded into the interpolator from the Parameter buffer~~
- ~~Q0 i, j, and k are loaded into the interpolator from the Pixel FIFO~~
- ~~The interpolator then generates the parameter value for each pixel in Q0 (Q0P0)~~
- ~~**P0i, P0j, and P0k are sent to the interpolator for Q1 only if Q1 is from a different primitive; if Q1 is from the same primitive as Q0, then the P0i, P0j, and P0k values loaded for Q0 are held by the interpolator and reused for Q1**~~
  - ~~**a "different_prim" control bit is passed with the barycentric data for each quad in the Pixel FIFO that indicates whether new parameter data needs to be loaded into the interpolator**~~
- ~~Q1 i, j, and k are then loaded into the interpolator from the Pixel FIFO~~
- ~~The interpolator then generates the parameter value for each pixel in Q1 (Q1P0)~~
- ~~Q2P0 and Q3P0 are generated in a similar manner~~
- ~~The next set of parameter data - P1i, P1j, and P1k - is then loaded into the interpolator~~
- ~~Q0 i, j, and k now must be re-read from the Pixel FIFO – this means that the output of the Pixel FIFO loops through the top four entries on each read command until at the end a final "block_pop" signal is asserted, causing the top four sets of barycentric coordinates to finally be removed~~
- ~~so the order of parameter info generated is Q0P0, Q1P0, Q2P0, Q3P0, Q0P1, Q1P1, etc.~~

~~7.~~5. SEQ controls the transfer of interpolated data to the SP register file over the RE_SP interface (which has a bandwidth of ~~512~~ 2048 bits/cycle). See interpolated data bus diagrams for details.

- ~~16 pixels worth of interpolated parameter data is sent to the 16 register files over 4 cycles~~
  - ~~RF0 of SU0, SU1, SU2, and SU3 is written with Q0P0 the first cycle~~
  - ~~RF1 of SU0, SU1, SU2, and SU3 is written with Q1P0 second cycle~~
  - ~~RF2 of SU0, SU1, SU2, and SU3 is written with Q2P0 third cycle~~
  - ~~RF3 of SU0, SU1, SU2, and SU3 is written with Q3P0 fourth cycle~~

~~8.~~6. SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of texture state machine 0, or TSM0 FIFO)

- note that there is a separate set of reservation stations/arbiters/state machines for vertices and for pixels
- the control packet contains the state pointer, the register file base pointer, and the LOD correction bits
- all other informations (such as quad address for example) travels in a separate FIFO

~~9.~~7. TSM0 accepts the control packet and fetches the instructions for texture clause 0 from the global instruction store

Formatted: Bullets and Numbering

- TSM0 was first selected by the TSM arbiter before it could start

~~10.~~8. all instructions of texture clause 0 are issued by TSM0

Formatted: Bullets and Numbering

~~11.~~9. the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)

- TSM0 does not wait for texture requests made to the Texture Unit to complete; it passes the register file write index for the texture data to the TU, which will write the data to the RF as it is received
- once the TU has written all the data for a particular clause to the register files, it increments a counter that is associated with the ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go ahead and pop the FIFO and start to execute the ALU clause

~~12.~~10. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store

Formatted: Bullets and Numbering

~~13.~~11. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of texture state machine 1, or TSM1 FIFO)

~~14.~~12. the control packet continues to travel down the path of reservation stations until all clauses have been executed

- pixel data is exported in the last ALU clause (clause 7)
  - it is sent to an output FIFO where it will be picked up by the render backend
  - the ASM arbiter will prevent a packet from starting on ASM7 if the output FIFO is full

~~15.~~13. after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program

Formatted: Bullets and Numbering

~~13.1.3~~15.1.3 *Notes*

Formatted: Bullets and Numbering

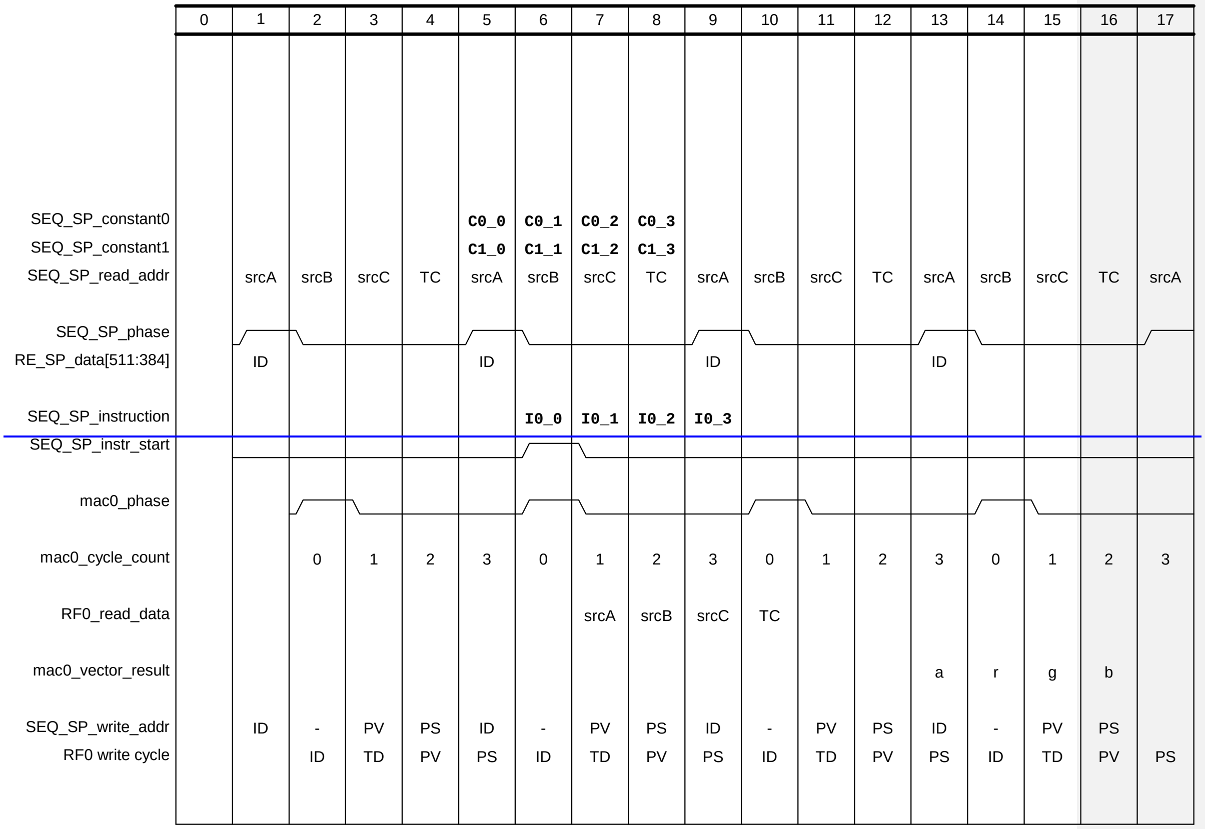
~~16.~~14. the state machines and arbitrers will operate ahead of time so that they will be able to immediately start the real threads or stall.

~~17.~~15. the register file base pointer for a vector needs to travel with the vector through the reservation stations, but the instruction store base pointer does not – this is because the RF pointer is different for all threads, but the IS pointer is only different for each state and thus can be accessed via the state pointer

16. Waterfalling, parameter buffer allocation, loops and branches and parameter cache de-allocation still needs to be specked out.

~~14.~~ Timing Diagrams

~~14.1~~ MAC 0

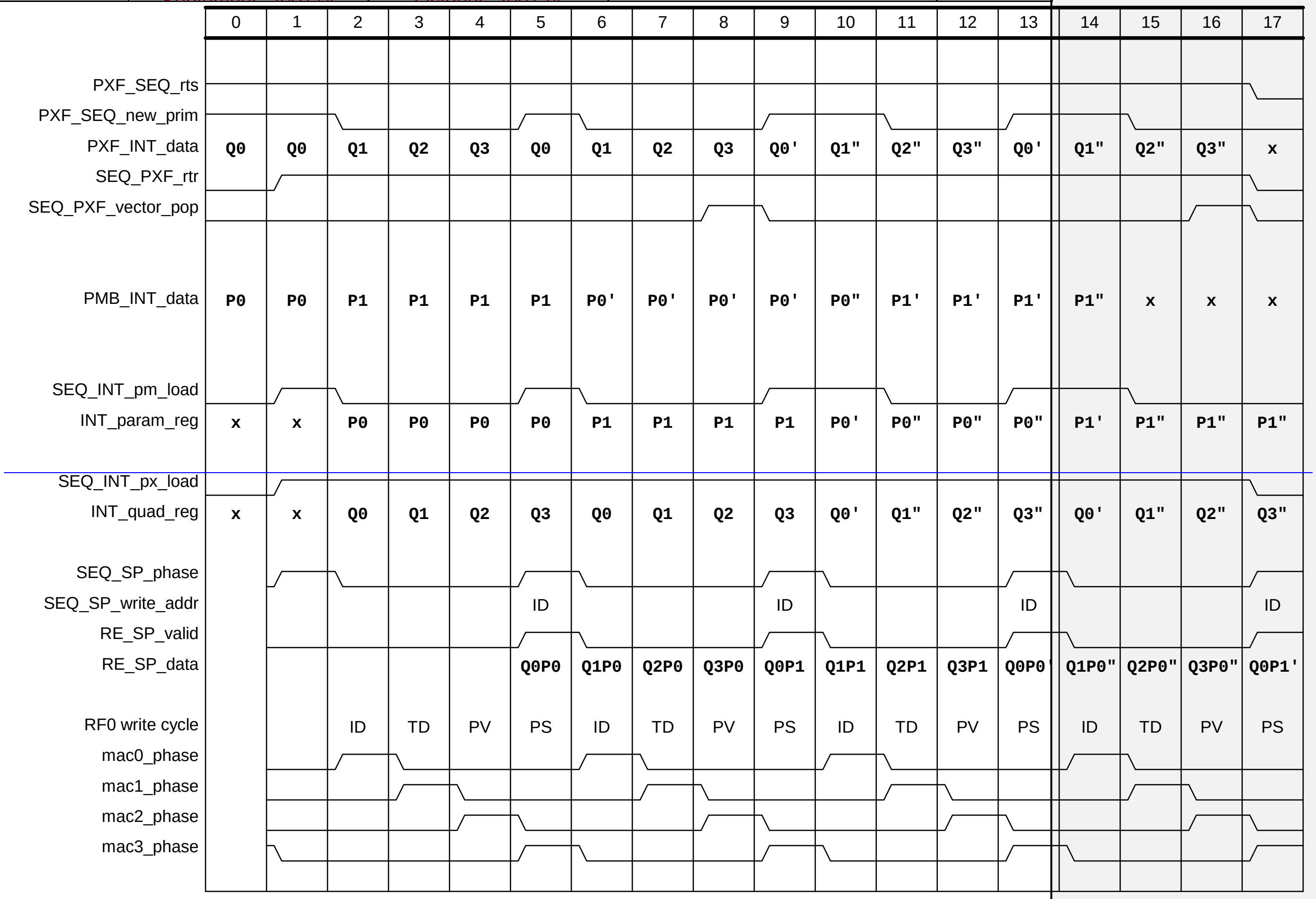


	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
SEQ_SP_constant0						C0_0	C0_1	C0_2	C0_3									
SEQ_SP_constant1						C1_0	C1_1	C1_2	C1_3									
SEQ_SP_read_addr		srcA	srcB	srcC	TC	srcA	srcB	srcC	TC	srcA	srcB	srcC	TC	srcA	srcB	srcC	TC	srcA
SEQ_SP_phase																		
RE_SP_data[511:384]		ID				ID				ID				ID				
SEQ_SP_instruction							I0_0	I0_1	I0_2	I0_3								
SEQ_SP_instr_start																		
mac0_phase																		
mac0_cycle_count			0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
RF0_read_data								srcA	srcB	srcC	TC							
mac0_vector_result														a	r	g	b	
SEQ_SP_write_addr		ID	-	PV	PS	ID	-	PV	PS	ID	-	PV	PS	ID	-	PV	PS	
RF0 write cycle			ID	TD	PV	PS	ID	TD	PV	PS	ID	TD	PV	PS	ID	TD	PV	PS

Timing Diagram 1: Sequencer to Shader Pipe 0, Shader Unit 0, MAC 0

~~14.2~~ Sequencer to Shader Pipe

Formatted: Bullets and Numbering

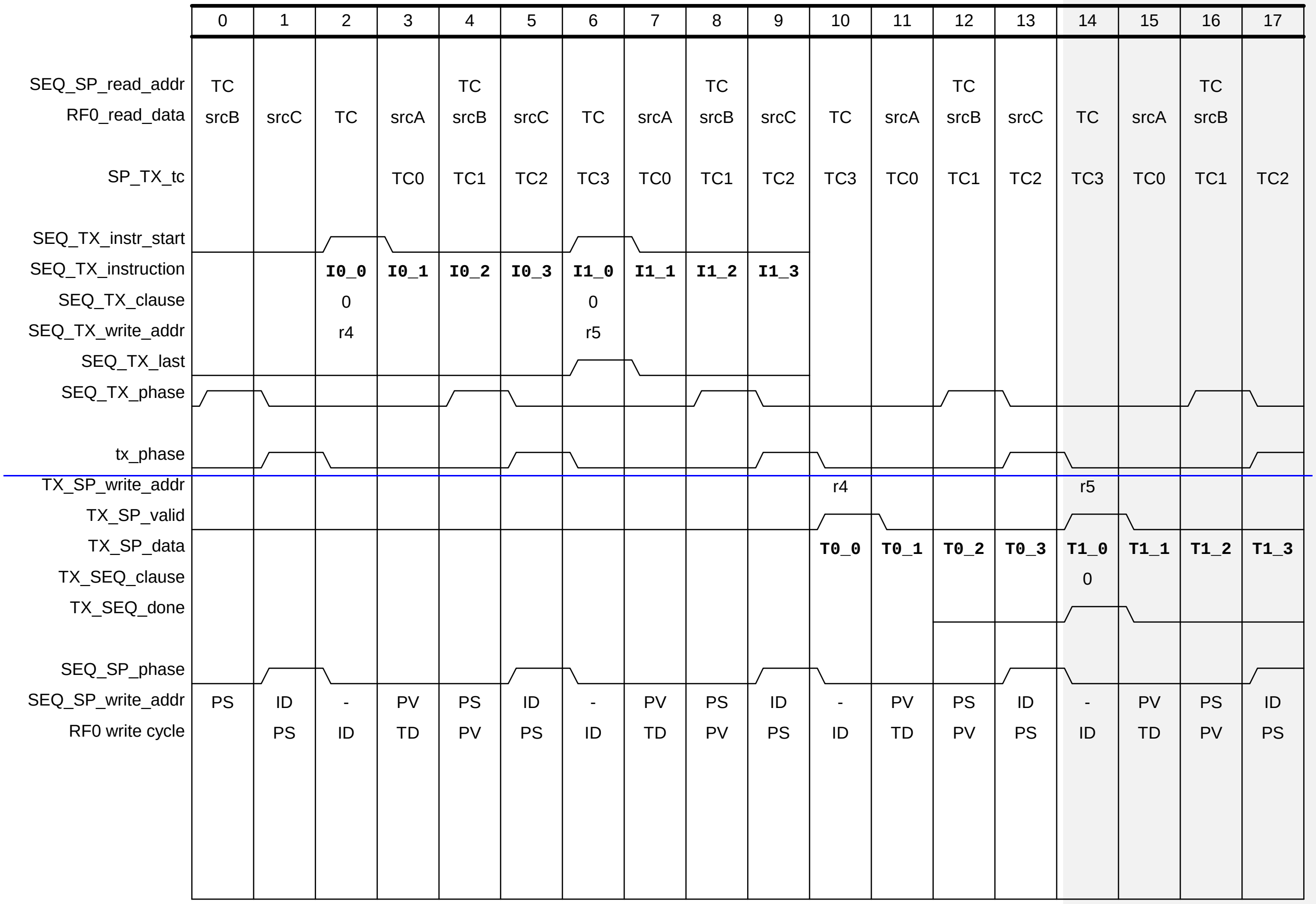


	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
PXF_SEQ_rts																		
PXF_SEQ_new_prim																		
PXF_INT_data	Q0	Q0	Q1	Q2	Q3	Q0	Q1	Q2	Q3	Q0'	Q1"	Q2"	Q3"	Q0'	Q1"	Q2"	Q3"	x
SEQ_PXF_rtr																		
SEQ_PXF_vector_pop																		
PMB_INT_data	P0	P0	P1	P1	P1	P1	P0'	P0'	P0'	P0'	P0"	P1'	P1'	P1'	P1"	x	x	x
SEQ_INT_pm_load																		
INT_param_reg	x	x	P0	P0	P0	P0	P1	P1	P1	P1	P0'	P0"	P0"	P0"	P1'	P1"	P1"	P1"
SEQ_INT_px_load																		
INT_quad_reg	x	x	Q0	Q1	Q2	Q3	Q0	Q1	Q2	Q3	Q0'	Q1"	Q2"	Q3"	Q0'	Q1"	Q2"	Q3"
SEQ_SP_phase																		
SEQ_SP_write_addr						ID				ID				ID				ID
RE_SP_valid																		
RE_SP_data						Q0P0	Q1P0	Q2P0	Q3P0	Q0P1	Q1P1	Q2P1	Q3P1	Q0P0	Q1P0"	Q2P0"	Q3P0"	Q0P1'
RF0 write cycle			ID	TD	PV	PS	ID	TD	PV	PS	ID	TD	PV	PS	ID	TD	PV	PS
mac0_phase																		
mac1_phase																		
mac2_phase																		
mac3_phase																		

Timing Diagram 2: RE Interpolator to Shader Pipe Data Transfer

~~14.3 Sequencer to Texture Pipe~~

Formatted: Bullets and Numbering



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
SEQ_SP_read_addr	TC				TC				TC				TC				TC	
RF0_read_data	srcB	srcC	TC	srcA	srcB	srcC	TC	srcA	srcB	srcC	TC	srcA	srcB	srcC	TC	srcA	srcB	
SP_TX_tc				TC0	TC1	TC2	TC3	TC0	TC1	TC2	TC3	TC0	TC1	TC2	TC3	TC0	TC1	TC2
SEQ_TX_instr_start																		
SEQ_TX_instruction			I0_0	I0_1	I0_2	I0_3	I1_0	I1_1	I1_2	I1_3								
SEQ_TX_clause			0				0											
SEQ_TX_write_addr			r4				r5											
SEQ_TX_last																		
SEQ_TX_phase																		
tx_phase																		
TX_SP_write_addr											r4				r5			
TX_SP_valid																		
TX_SP_data											T0_0	T0_1	T0_2	T0_3	T1_0	T1_1	T1_2	T1_3
TX_SEQ_clause															0			
TX_SEQ_done																		
SEQ_SP_phase																		
SEQ_SP_write_addr	PS	ID	-	PV	PS	ID	-	PV	PS	ID	-	PV	PS	ID	-	PV	PS	ID
RF0 write cycle		PS	ID	TD	PV	PS	ID	TD	PV	PS	ID	TD	PV	PS	ID	TD	PV	PS

Timing Diagram 3: Sequencer - Texture Unit Interface and Texture Unit - Shader Pipe Data Transfer

~~14.4 Timing diagrams explanations~~

Formatted: Bullets and Numbering

~~The numbering of the four shader pipes, the four shader units, and the four MACs is from left to right and from 0 to 3. So for example the most significant 512 bits of a SP goes to SU0 and the least significant 512 bits go to SU3; within SU0, the most significant 128 bits go to MAC0 and the least significant 128 bits go to MAC3.~~
~~The following assumptions are made:~~

~~1.all block to block signals are register to register~~

Formatted: Bullets and Numbering

~~2.for register file reads, the RF read data is available in the MAC one clock after a RF read address is registered into the MAC (this is the same as saying the read data is valid out of the RF two clocks after the address is asserted on the SEQ to SP interface)~~

~~14.4.1 *Timing Diagram 1: Sequencer to Shader Pipe 0, Shader Unit 0, MAC 0*~~

~~This diagram shows the basics of the Sequencer to Shader Pipe interface. For simplicity only the timing relative to MAC0 is shown. The timing for MAC1 is one clock later than MAC0, MAC2 one clock later than MAC1, etc. This means that most of the signals need to be delayed in the SP by one cycle for MAC1, two cycles for MAC2, and three cycles for MAC3.~~
~~**SEQ_SP_constant0**: Constant 0 (128 bits over 4 cycles). Pipelined in SP for other MACs.~~
~~**SEQ_SP_constant1**: Constant 1 (128 bits over 4 cycles). Pipelined in SP for other MACs.~~
~~**SEQ_SP_read_addr**: Register File Read Address (8 bits). Pipelined in SP for other MACs.~~
~~**SEQ_SP_phase**: This signal syncs the data transfer to the RF from the RE, as well as defining the order of all writes into the RF. It is asserted during the cycle that interpolated data (ID) is valid on the RE_SP_ID bus. Pipelined in SP for other MACs.~~
~~**RE_SP_ID[511:384]**: This is the most significant 128 bits of the RE_SP_data interface (meaning that this MAC0 is in SU0).~~
~~**SEQ_SP_instruction**: 96 bits of instruction are sent over 4 cycles. Pipelined in SP for other MACs.~~
~~**SEQ_SP_instr_start**: control bit that signals the first cycle of the instruction transfer. Pipelined in SP for other MACs.~~
~~**mac0_phase**: registered version of SEQ_SP_phase used in MAC0 (this may not be he actual signal name).~~
~~**mac0_cycle_count**: a counter inside the MAC that keeps track of the RF write cycles; 0 here corresponds to the cycle RE interpolated data is written (this may not be he actual signal name).~~
~~**RF0_read_data**: data that is read out of MAC0's register file (this may not be he actual signal name).~~
~~**mac0_vector_result**: the 32-bit output of the vector ALU (PV is built up over 4 cycles) (this may not be he actual signal name).~~
~~**SEQ_SP_write_addr**: Register File Write Address (8 bits). Note that the SEQ does not send the Texture Data write address over this bus. Pipelined in SP for other MACs.~~
~~**RF0 write cycle**: the cycles allocated to the different write sources (ID = Interpolated Data, TD = Texture Data, PV = Previous Vector, PS = Previous Scalar) (not a signal – just a reference point on the diagram).~~

~~14.4.2 *Timing Diagram 2: RE Interpolator to Shader Pipe Data Transfer*~~

Formatted: Bullets and Numbering

~~This diagram shows how pixel data (barycentric coordinates i, j, and k) is sent from the Pixel FIFO to the interpolator under SEQ control, and how parameter data (for each vertex) is also sent to the interpolator under SEQ control. The output of the Interpolator is then shown being sent over the RE_SP interface.~~
~~**PXF_SEQ_rts**: Indicates that the output of the pixel FIFO is valid.~~
~~**PXF_SEQ_new_prim**: The current output of the Pixel FIFO is from a different primitive that the previous output. Tells the SEQ that new parameter info must be fetched (if its not from a new prim, then new parameter data is not needed).~~
~~**PXF_INT_data**: Data output of the Pixel FIFO – goes to the Interpolator.~~
~~**SEQ_PXF_rtr**: Indicates that the current Pixel FIFO output will be taken by the Interpolator (driven by SEQ). Then next quad of data will be driven the next cycle.~~
~~**SEQ_PXF_vector_pop**: SEQ tells the Pixel FIFO to pop a vector of pixels (otherwise RTRs cause the data to be cycled between the four quads).~~

~~**PMB_INT_data**: Data from the Parameter Buffer to the Interpolator. (Note that the control of the parameter buffer is TBD).~~

~~**SEQ_INT_pm_load**: controls the loading of parameter data into the Interpolator.~~

~~**INT_param_reg**: register in the Interpolator that holds the per-vertex parameter data while the per-pixel parameters are generated for one or more quads (may not be the actual signal name).~~
~~**SEQ_INT_px_load**: controls the loading of pixel data into the Interpolator.~~
~~**INT_quad_reg**: : register in the Interpolator that holds one quad's worth of pixel data(may not be the actual signal name).~~

~~**SEQ_SP_phase**: see above under TD1.~~
~~**SEQ_SP_write_addr**: see above under TD1.~~
~~**RE_SP_valid**: Interpolator Data Valid – indicates that the SP should write the ID on the appropriate cycle.~~
~~**RE_SP_data**: Data from the RE interpolator to the SP.~~
~~**RF0 write cycle**: see above under TD1.~~
~~**mac*_phase**: see above under TD1. These phase signals help to show the timing offset between the MACs. Note also that each Shader Unit has a set of these signals (all with the same timing).~~

~~14.4.3 *Timing Diagram 3: Sequencer - Texture Unit Interface and Texture Unit - Transfer*~~

Formatted: Bullets and Numbering

~~This diagram starts with the texture coordinate read from the register file and its transfer to the TX.   The instruction transfer is then shown, followed by the texture data transfer to the shader pipe.~~
~~**SEQ_SP_read_addr**: see above. Here shows the cycle that the texture coordinate read address is asserted.~~
~~**RF0_read_addr**: see above.~~
~~**SP_TX_tc**: Texture coordinate data sent from the shader pipe to the texture unit.~~
~~**SEQ_TX_instr_start**: Asserted on the first cycle of a SEQ to TX instruction transfer.~~
~~**SEQ_TX_instruction**: 96 bits of texture instruction transferred over 4 cycles.~~
~~**SEQ_TX_clause**: the clause number associated with this instruction.~~
~~**SEQ_TX_write_addr**: RF write index used by TX for returned texture data.~~
~~**SEQ_TX_last**: indicates that this is the last texture instruction of a clause.~~
~~**SEQ_TX_phase**: syncs the texture data write. Note that it is asserted early enough to be registered into TX and still allow TX to source the texture data to the SP on the correct cycle.~~

~~**tx_phase**: the phase signal after being registered into TX.~~
~~**TX_SP_write_addr**: RF write index for texture data.~~
~~**TX_SP_valid**: indicates that valid texture data is being driven to the SP.~~
~~**TX_SP_data**: the texture data.~~
~~**TX_SEQ_clause**: the clause number associated with the texture data.~~
~~**TX_SEQ_done**: indicates to the SEQ that the texture data transfer is complete for the clause number that is on the TX_SEQ_clause bus.~~

~~**SEQ_SP_phase**: see above under TD1 - shown here for reference.~~
~~**SEQ_SP_write_addr**: see above under TD1- shown here for reference.~~
~~**RF0 write cycle**: see above under TD1- shown here for reference.~~

~~15.~~16. Open issues

Formatted: Bullets and Numbering

There is currently an issue with constants. If the constants are not the same for the whole vector of vertices, we don't have the bandwith from the texture store to feed the ALUs. Two solutions exists for this problem:

1) Let the compiler handle the case and put those instructions in a texture clause so we can use the bandwith there to operate. This requires a significant amount of temporary storage in the register store.
2) Waterfall down the pipe allowing only at a given time the vertices having the same constants to operate in parrallel. This might in the worst case slow us down by a factor of 16.

Need to do some testing on the size of the register file as well as on the register file allocation method (dynamic VS static).

Saving power?

Size of the fifo containing the information of a vector of pixels/vertices. And size of the fifos before the reservation stations.

Sequencer Instruction memory, and constant memory.

Arbitration policy for the output file.

Loops and branches.

The parameter cache may end up in the PA rather than in the RS. Parameter cache management thus may change.

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001~~24~~ September, 200114	4 September, 2015~~5~~ October, 200124	GEN-CXXXXX-REVA	1 of 33

Author: Laurent Lefebvre

Issue To:	**Copy No:**

R400 Sequencer Specification

SEQ

Version 0.7~~5~~

Overview: This is an architectural specification for the R400 Sequencer block (SEQ). It provides an overview of the required capabilities and expected uses of the block. It also describes the block interfaces, internal sub-blocks, and provides internal state diagrams.

AUTOMATICALLY UPDATED FIELDS:
Document Location: C:\perforce\r400\arch\doc\gfx\RE\R400_Sequencer.doc
Current Intranet Search Title: R400 Sequencer Specification

APPROVALS	
Name/Dept	Signature/Date

Remarks:

THIS DOCUMENT CONTAINS CONFIDENTIAL INFORMATION THAT COULD BE SUBSTANTIALLY DETRIMENTAL TO THE INTEREST OF ATI TECHNOLOGIES INC. THROUGH UNAUTHORIZED USE OR DISCLOSURE.

Table Of Contents

1. OVERVIEW .... 5~~4~~
1.1 Top Level Block Diagram .... 7~~6~~
1.2 Data Flow graph .... 10~~8~~
1.3 Control Graph.................................... 12~~9~~
2. INTERPOLATED DATA BUS......... 12~~9~~
3. INSTRUCTION STORE .... 15~~12~~
4. SEQUENCER INSTRUCTIONS.... 15~~12~~
5. CONSTANT STORE..................... 15~~12~~
6. LOOPING AND BRANCHES........ 15~~12~~
6.1 The controlling state. .... 15~~12~~
6.2 The Control Flow Program .... 16~~13~~
6.3 Data dependant predicate instructions 17~~14~~
6.4 Register file indexing..................... 18~~14~~
7. REGISTER FILE ALLOCATION... 18~~15~~
8. TEXTURE ARBITRATION............ 19~~16~~
9. ALU ARBITRATION .... 19~~16~~
10. HANDLING STALLS .... 20~~17~~
11. CONTENT OF THE RESERVATION STATION FIFOS.................................... 20~~17~~
12. THE OUTPUT FILE....................... 20~~17~~
13. THE PARAMETER CACHE.......... 21~~17~~
14. VERTEX POSITION EXPORTING 21~~17~~
15. REGISTERS .... 21~~17~~
16. INTERFACES......................... 22~~18~~
16.1 External Interfaces............. 22~~18~~
16.1.1 PA/SC to RE : IJ bus..... 22~~18~~
16.1.2 PA/SC to SEQ : IJ Control bus 22~~18~~
16.1.3 PA/SC to RE : Vertex Bus. 23~~18~~
16.1.4 PA/SC to SEQ : Vertex Control Bus 23~~18~~
16.1.5 CP to SEQ : Constant store load 23~~19~~
16.1.6 CP to SEQ : Texture State store load 23~~19~~
16.1.7 CP to SEQ : Control State store load 23~~19~~
16.1.8 MH to SEQ: Instruction store Load 23~~19~~
16.1.9 SP to RB : Pixel read from RBs 24~~19~~
16.1.10 SP to PA/SC : Position return bus 24~~19~~
16.1.11 Shader Engine to Texture Unit Bus (Fast Bus) .... 24~~19~~
16.1.12 Sequencer to Texture Unit bus (Slow Bus) 24~~20~~
17. INTERNAL INTERFACES .... 25~~20~~
18. EXAMPLES OF PROGRAM EXECUTIONS... 25~~20~~
18.1.1 Sequencer Control of a Vector of Vertices 25~~20~~
18.1.2 Sequencer Control of a Vector of Pixels 26~~21~~
18.1.3 Notes.................................. 27~~22~~
19. OPEN ISSUES .... 33~~23~~
~~**1. OVERVIEW.. 3**~~
~~1.1 Top Level Block Diagram .... 5~~
~~1.2 Data Flow graph.................................. 7~~
~~1.3 Control Graph...................................... 8~~
~~**2. INTERPOLATED DATA BUS............. 8**~~
~~**3. INSTRUCTION STORE .... 11**~~
~~**4. SEQUENCER INSTRUCTIONS........ 11**~~
~~**5. CONSTANT STORE .... 11**~~
~~**6. LOOPING AND BRANCHES............ 11**~~
~~6.1 Data dependant predicate instructions 11~~
~~6.2 Register file indexing.......................... 12~~
~~**7. REGISTER FILE ALLOCATION....... 12**~~
~~**8. TEXTURE ARBITRATION................ 13**~~
~~**9. ALU ARBITRATION .... 13**~~
~~**10. HANDLING STALLS .... 14**~~
~~**11. CONTENT OF THE RESERVATION STATION FIFOS.. 14**~~
~~**12. THE OUTPUT FILE (RB FIFO AND PARAMETER CACHE)................................ 14**~~
~~**13. REGISTERS .... 14**~~
~~**14. INTERFACES................................ 14**~~
~~14.1 External Interfaces................. 14~~
~~14.1.1 Sequencer to Shader Engine Bus... 14~~
~~14.1.2 Shader Engine to Output File 15~~

~~14.1.3 Shader Engine to Texture Unit Bus (Fast Bus) .... 15~~
~~14.1.4 Sequencer to Texture Unit bus (Slow Bus) 15~~
~~14.1.5 Shader Engine to RE/PA Bus 15~~
~~14.1.6 PA? to sequencer.... 16~~
~~15. EXAMPLES OF PROGRAM EXECUTIONS.... 16~~
~~15.1.1 Sequencer Control of a Vector of Vertices 16~~
~~15.1.2 Sequencer Control of a Vector of Pixels 17~~
~~15.1.3 Notes.... 18~~
~~16. OPEN ISSUES.... 19~~
~~1. OVERVIEW.... 3~~
~~1.1 Top Level Block Diagram.... 5~~
~~1.2 Data Flow graph.... 7~~
~~1.3 Control Graph.... 10~~
~~2. INTERPOLATED DATA BUS.... 10~~
~~3. INSTRUCTION STORE.... 13~~
~~4. CONSTANT STORE.... 13~~
~~5. LOOPING AND BRANCHES.... 13~~
~~6. REGISTER FILE ALLOCATION.... 13~~
~~7. TEXTURE ARBITRATION.... 14~~
~~8. ALU ARBITRATION.... 14~~
~~9. HANDLING STALLS.... 15~~
~~10. CONTENT OF THE RESERVATION STATION FIFOS.... 15~~
~~11. THE OUTPUT FILE (RB FIFO AND PARAMETER CACHE).... 15~~
~~12. INTERFACES.... 15~~
~~12.1 External Interfaces.... 15~~
~~12.1.1 Sequencer to Shader Engine Bus.... 15~~
~~12.1.2 Shader Engine to Output File 15~~
~~12.1.3 Shader Engine to Texture Unit Bus (Fast Bus).... 16~~
~~12.1.4 Sequencer to Texture Unit bus (Slow Bus) 16~~
~~12.1.5 Shader Engine to RE/PA Bus 16~~
~~12.1.6 PA? to sequencer.... 16~~
~~13. EXAMPLES OF PROGRAM EXECUTIONS.... 16~~
~~13.1.1 Sequencer Control of a Vector of Vertices 16~~
~~13.1.2 Sequencer Control of a Vector of Pixels 18~~
~~13.1.3 Notes.... 18~~
~~14. OPEN ISSUES.... 20~~
~~1. OVERVIEW.... 3~~
~~1.1 Top Level Block Diagram.... 4~~
~~1.2 Data Flow graph.... 7~~
~~1.3 Control Graph.... 10~~
~~2. INTERPOLATED DATA BUS.... 10~~
~~3. INSTRUCTION STORE.... 10~~
~~4. CONSTANT STORE.... 11~~
~~5. LOOPING AND BRANCHES.... 11~~
~~6. REGISTER FILE ALLOCATION.... 11~~
~~7. TEXTURE ARBITRATION.... 12~~
~~8. ALU ARBITRATION.... 12~~
~~9. HANDLING STALLS.... 13~~
~~10. CONTENT OF THE RESERVATION STATION FIFOS.... 13~~
~~11. THE OUTPUT FILE (RB FIFO AND PARAMETER CACHE).... 13~~
~~12. INTERFACES.... 13~~
~~12.1 External Interfaces.... 13~~
~~12.1.1 Sequencer to Shader Engine Bus.... 13~~
~~12.1.2 Shader Engine to Output File 13~~
~~12.1.3 Shader Engine to Texture Unit Bus (Fast Bus).... 14~~
~~12.1.4 Sequencer to Texture Unit bus (Slow Bus) 14~~
~~12.1.5 Shader Engine to RE/PA Bus 14~~
~~12.1.6 PA? to sequencer.... 14~~
~~13. EXAMPLES OF PROGRAM EXECUTIONS.... 14~~
~~13.1.1 Sequencer Control of a Vector of Vertices 14~~
~~13.1.2 Sequencer Control of a Vector of Pixels 16~~
~~13.1.3 Notes.... 17~~
~~14. OPEN ISSUES.... 17~~

Revision Changes:

Rev 0.1 (Laurent Lefebvre) Date: May 7, 2001	First draft.
Rev 0.2 (Laurent Lefebvre) Date : July 9, 2001	Changed the interfaces to reflect the changes in the SP. Added some details in the arbitration section.
Rev 0.3 (Laurent Lefebvre) Date : August 6, 2001	Reviewed the Sequencer spec after the meeting on August 3, 2001.
Rev 0.4 (Laurent Lefebvre) Date : August 24, 2001	Added the dynamic allocation method for register file and an example (written in part by Vic) of the flow of pixels/vertices in the sequencer.
Rev 0.~~4~~ 5 (Laurent Lefebvre) Date : September 7, 2001	Added timing diagrams (Vic)
Rev 0.6 (Laurent Lefebvre) Date : September 24, 2001	Changed the spec to reflect the new R400 architecture. Added interfaces.
Rev 0.7 (Laurent Lefebvre) Date : October 5, 2001	Added constant store management, instruction store management, control flow management and data dependant predication.

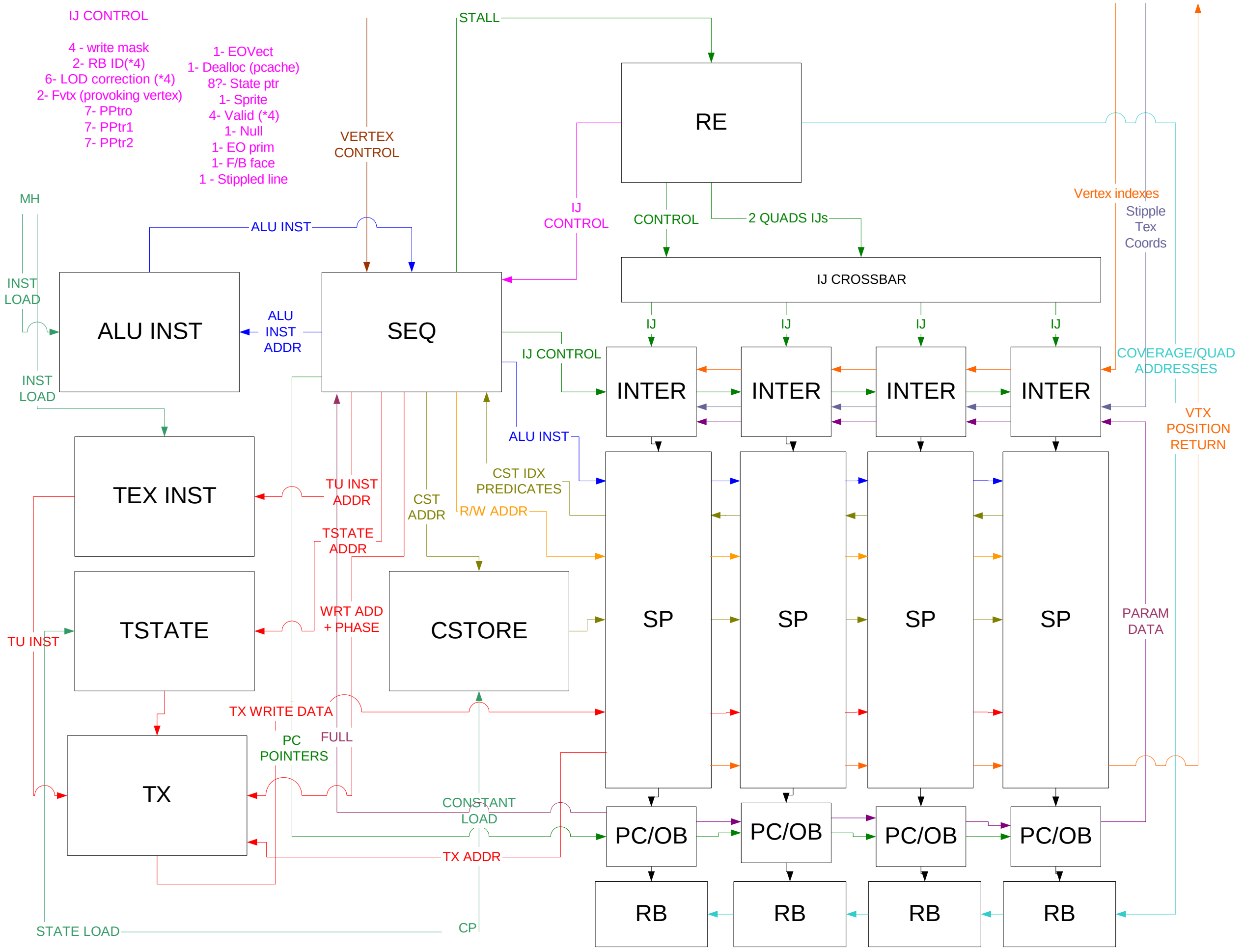
1. Overview

The sequencer first arbitrates between vectors of ~~16~~ 64 vertices that arrive directly from primitive assembly and vectors of ~~4~~ 16 quads (~~16~~ 64 pixels) that are generated in the raster engine.

The vertex or pixel program specifies how many GPR's it needs to execute. The sequencer will not start the next vector until the needed space is available.

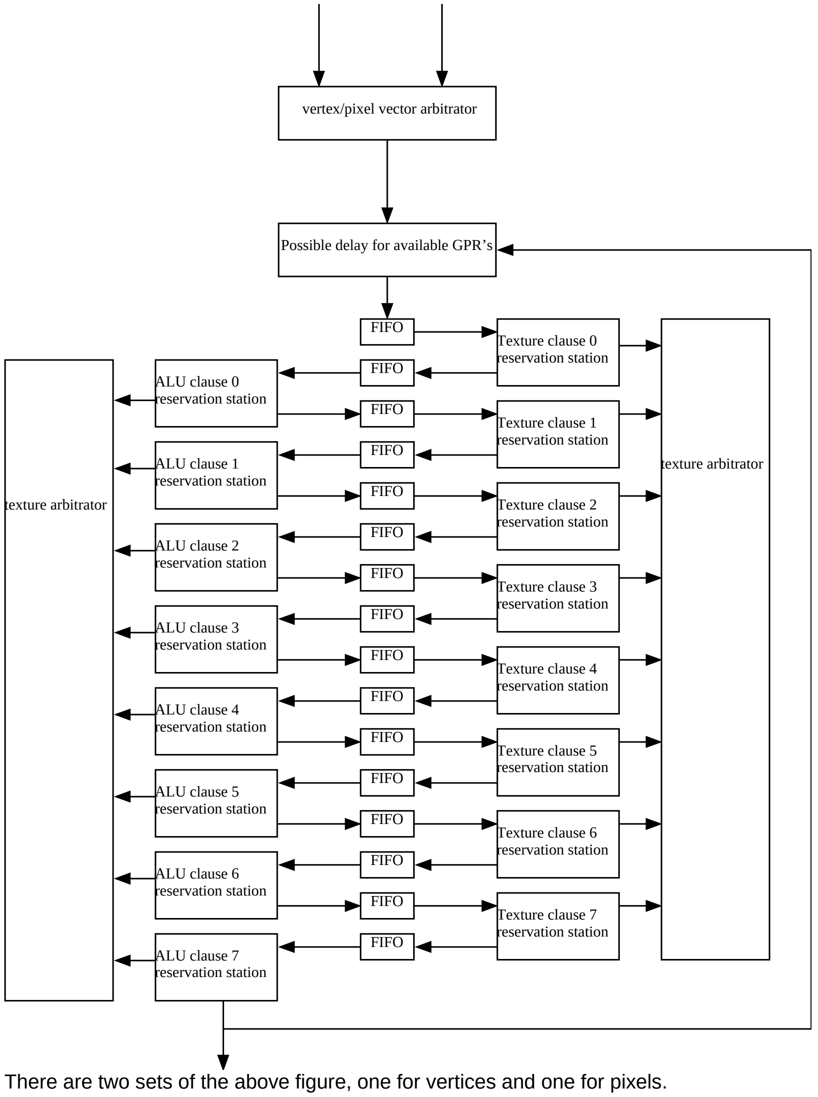
The sequencer is based on the R300 design. It chooses two ALU clauses and a texture clause to execute, and executes all of the instructions in a clause before looking for a new clause of the same type. Two ALU clauses are executed interleaved to hide the ALU latency. Each vector will have eight texture and eight ALU clauses, but clauses do not need to contain instructions. A vector of pixels or vertices ping-pongs along the sequencer FIFO, bouncing from texture reservation station to alu reservation station. A FIFO exists between each reservation stage, holding up vectors until the vector currently occupying a reservation station has left. A vector at a reservation station can be chosen to execute. The sequencer looks at all eight alu reservation stations to choose an alu clause to execute and all eight texture stations to choose a texture clause to execute. The arbitrator will give priority to clauses/reservation stations closer to the bottom of the pipeline. It will not execute an alu clause until the texture fetches initiated by the previous texture clause have completed. There are two separate sets of reservation stations, one for pixel vectors and one for vertices vectors. This way a pixel can pass a vertex and a vertex can pass a pixel.

To support the shader pipe the raster engine also contains the shader instruction cache and constant store. There are only one constant store for the whole chip and one instruction store. These will be shared among the four shader pipes. The four shader pipes also execute the same instuction thus there is only one sequencer for the whole chip.



IJ CONTROL
4 - write mask
2- RB ID(*4)
6- LOD correction (*4)
2- Fvtx (provoking vertex)
7- PPtro
7- PPtr1
7- PPtr2
1- EOVect
1- Dealloc (pcache)
8?- State ptr
1- Sprite
4- Valid (*4)
1- Null
1- EO prim
1- F/B face
1 - Stippled line
VERTEX CONTROL
STALL
RE
IJ CONTROL
CONTROL
2 QUADS IJs
Vertex indexes
Stipple Tex Coords
MH
ALU INST
INST LOAD
ALU INST
SEQ
ALU INST ADDR
IJ CROSSBAR
IJ
IJ
IJ
IJ
IJ CONTROL
INTER
INTER
INTER
INTER
COVERAGE/QUAD ADDRESSES
VTX POSITION RETURN
INST LOAD
ALU INST
TEX INST
TU INST ADDR
CST ADDR
CST IDX PREDICATES
R/W ADDR
TSTATE ADDR
SP
SP
SP
SP
PARAM DATA
TU INST
TSTATE
WRT ADD + PHASE
CSTORE
TX WRITE DATA
FULL
PC POINTERS
TX
CONSTANT LOAD
PC/OB
PC/OB
PC/OB
PC/OB
TX ADDR
RB
RB
RB
RB
STATE LOAD
CP

1.1 Top Level Block Diagram



vertex/pixel vector arbitrator
Possible delay for available GPR's
FIFO
Texture clause 0 reservation station
Texture clause 1 reservation station
Texture clause 2 reservation station
Texture clause 3 reservation station
Texture clause 4 reservation station
Texture clause 5 reservation station
Texture clause 6 reservation station
Texture clause 7 reservation station
texture arbitrator
ALU clause 0 reservation station
ALU clause 1 reservation station
ALU clause 2 reservation station
ALU clause 3 reservation station
ALU clause 4 reservation station
ALU clause 5 reservation station
ALU clause 6 reservation station
ALU clause 7 reservation station
texture arbitrator

There are two sets of the above figure, one for vertices and one for pixels.

The rasterizer always checks the vertices FIFO first and if allowed by the sequencer sends the data to the shader. If the vertex FIFO is empty then, the rasterizer takes the first entry of the pixel FIFO (a vector of ~~16~~ 64 pixels) and sends it to the interpolators. Then the sequencer takes control of the packet. The packet consists of ~~3~~ 21 bits of state, 6-7 bits for the base address of the Shader program and some information on the coverage to determine texture LOD. All other information (2x2 adresses) is put in a FIFO (one for the pixels and one for the vertices) and retrieved when the packet finishes its last clause.

On receipt of a packet, the input state machine (not pictured but just before the first FIFO) allocated enough space in the registers to store the interpolated values and temporaries. Following this, the input state machine stacks the packet in the first FIFO.

On receipt of a command, the level 0 texture machine issues a texure request and corresponding register address for the texture address (ta). A small command (tcmd) is passed to the texture system identifying the current level number (0) as well as the register write address for the texture return data. One texture request is sent every 4 clocks causing the texturing of ~~four~~ sixteen 2x2s worth of data (or ~~16~~ 64 vertices). Once all the requests are sent the packet is put in FIFO 1.

Upon recept of the return data, the texture unit writes the data to the register file using the write address that was provided by the level 0 texture machine and sends the clause number (0) to the level 0 texture state machine to signify that the write is done and thus the data is ready. Then, the level 0 texture machine increments the counter of FIFO 1 to signify to the ALU 1 that the data is ready to be processed.

On receipt of a command, the level 0 ALU machine first decrements the input FIFO counter and then issues a complete set of level 0 shader instructions. For each instruction, the state machine generates 3 source addresses, one destination address (3 cycles later) and an instruction. Once the last instruction as been issued, the packet is put into FIFO 2.

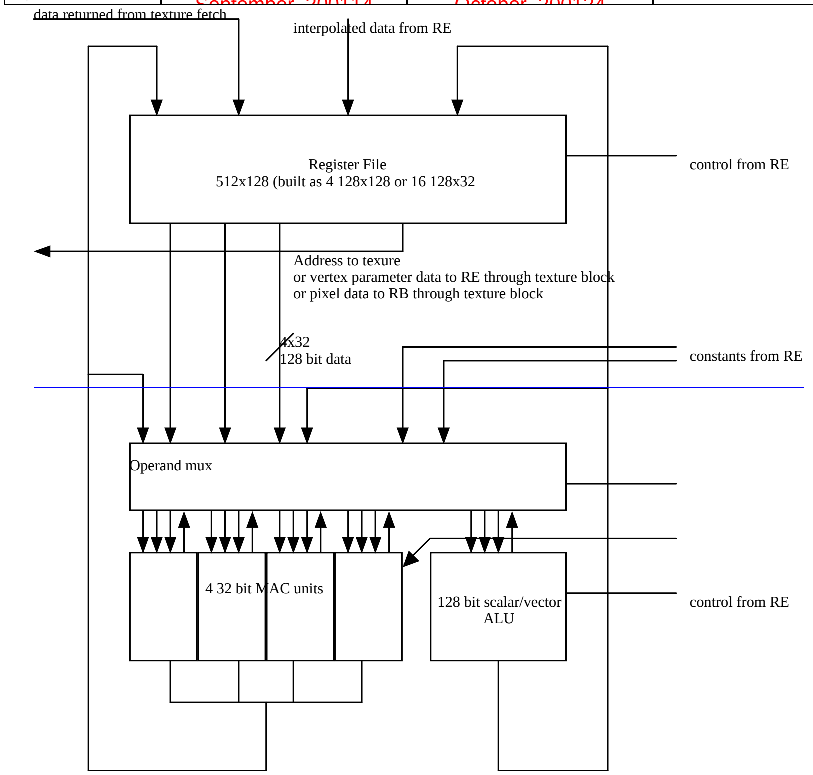
There will always be two active ALU clauses at any given time (and two arbitrers). One arbitrer will arbitrate over the odd clock cycles and the other one will arbitrate over the even clock cycles. The only constraints between the two arbitrers is that they are not allowed to pick the same clause number as they other one is currently working on if the packet os of the same type.

If the packet is a vertex packet, upon reaching ALU clause 4, it can export the position if the position is ready. So the arbitrer must prevent ALU clause 4 to be selected if the positional buffer is full (or can't be accessed). Along with the positional data, the location where the vertex data is to be put is also sent (parameter data pointers).

All other level process in the same way until the packet finally reaches the last ALU machine (8). On completion of the level 8 ALU clause, a valid bit is sent to the Render Backend which picks up the color data. This requires that the last instruction writes to the output register – a condition that is almost always true. If the packet was a vertex packet, instead of sending the valid bit to the RB, it is sent to the PA so it can know that the data present in the parameter store is valid.

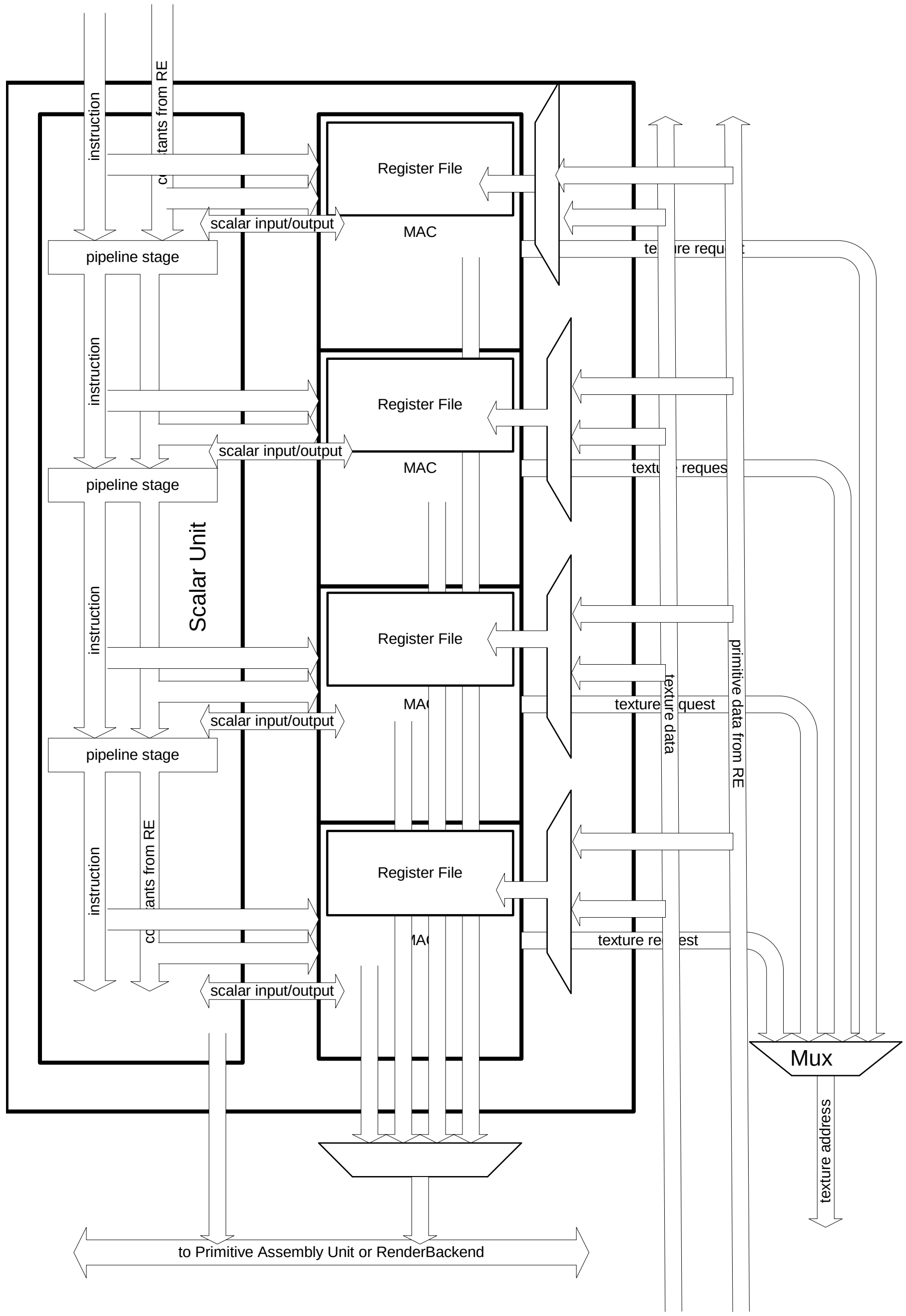
Only two ALU state machine may have access to the register file address bus or the instruction decode bus at one time. Similarly, only one texture state machine may have access to the register file address bus at one time. Arbitration is performed by three arbitrer blocks (two for the ALU state machines and one for the texture state machines). The arbitrers always favor the higher number state machines, preventing a bunch of half finished jobs from clogging up the register files.

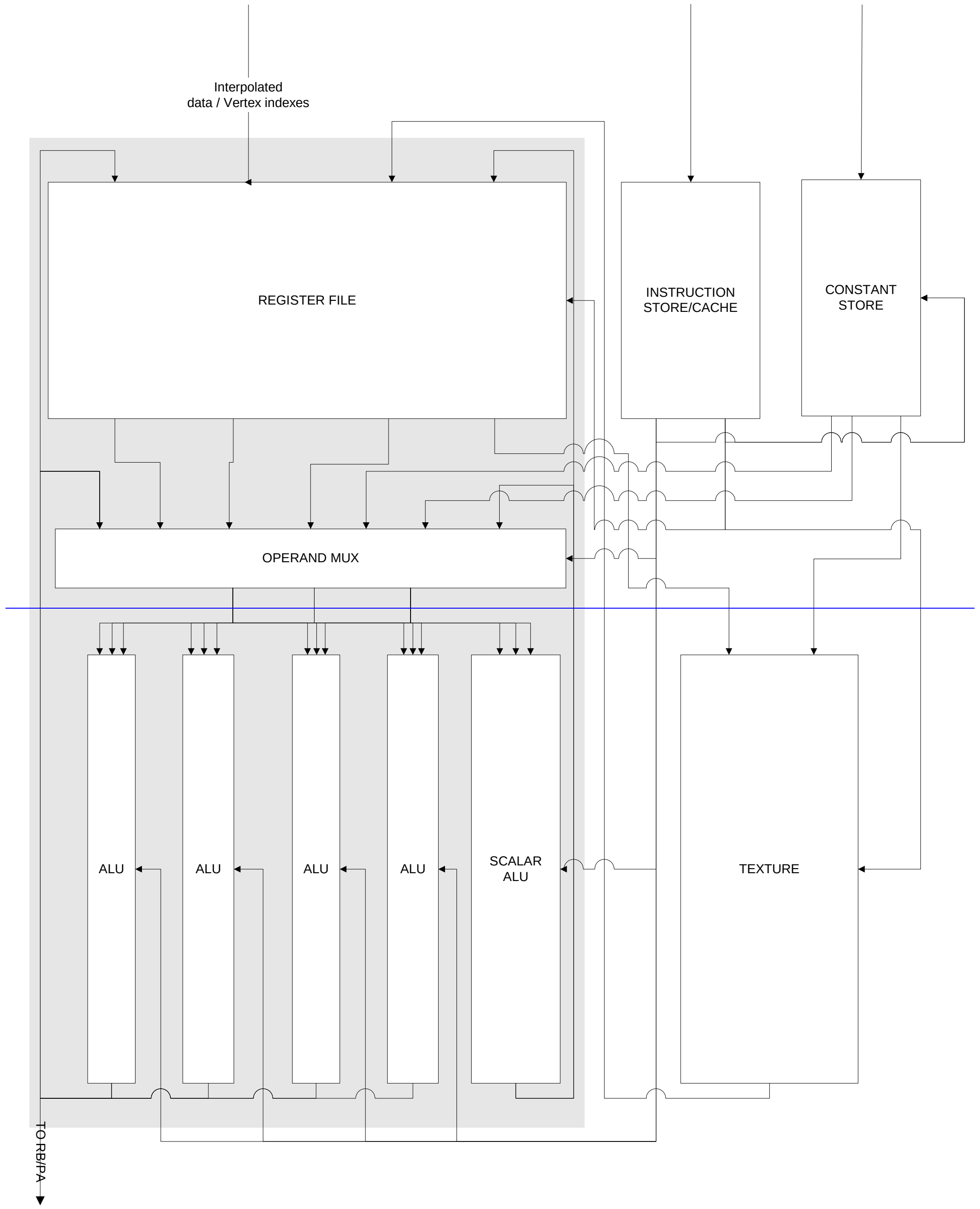
~~Each state machine maintains an address pointer specifying where the 16 entries vector is located in the register file (the texture machine has two pointers one for the read address and one for the write). Upon completion of its job, the address pointer is incremented by a predefined amount equal to the total number of registers required by the shading code. A comparison of the address pointer for the first state machine in the chain (the input state machine), and the last machine in the chain (the level 8 ALU machine), gives an indication of how much unallocated register file memory is available~~



data returned from texture fetch
interpolated data from RE
Register File
512x128 (built as 4 128x128 or 16 128x32
control from RE
Address to texure
or vertex parameter data to RE through texture block
or pixel data to RB through texture block
4x32
128 bit data
constants from RE
Operand mux
4 32 bit MAC units
128 bit scalar/vector ALU
control from RE

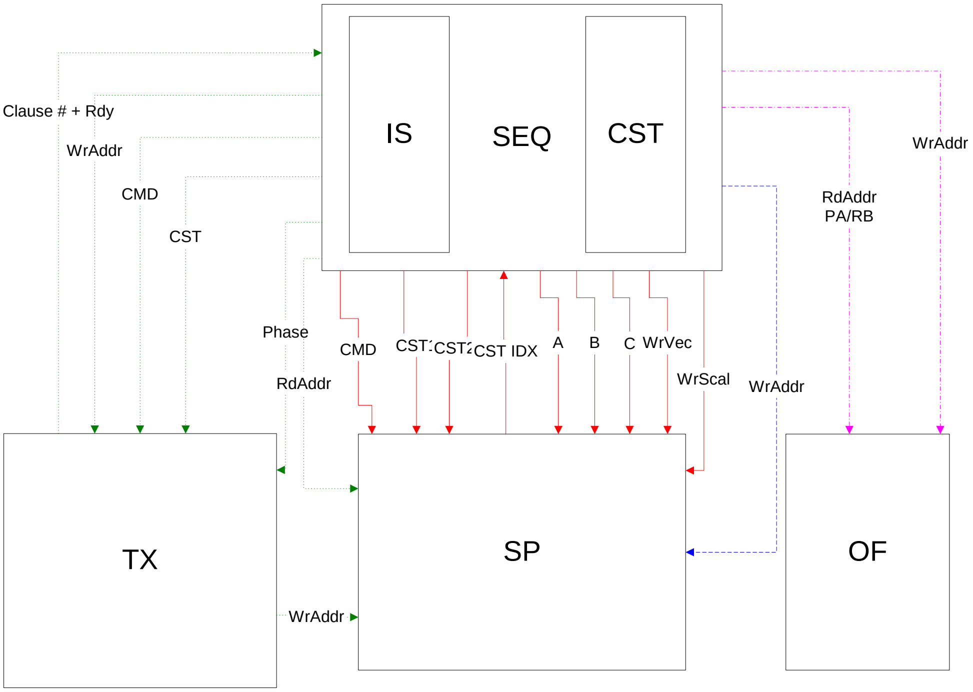
1.2 Data Flow graph





The gray area represents blocks that are replicated 4 times per shader pipe (16 times on the overall chip).

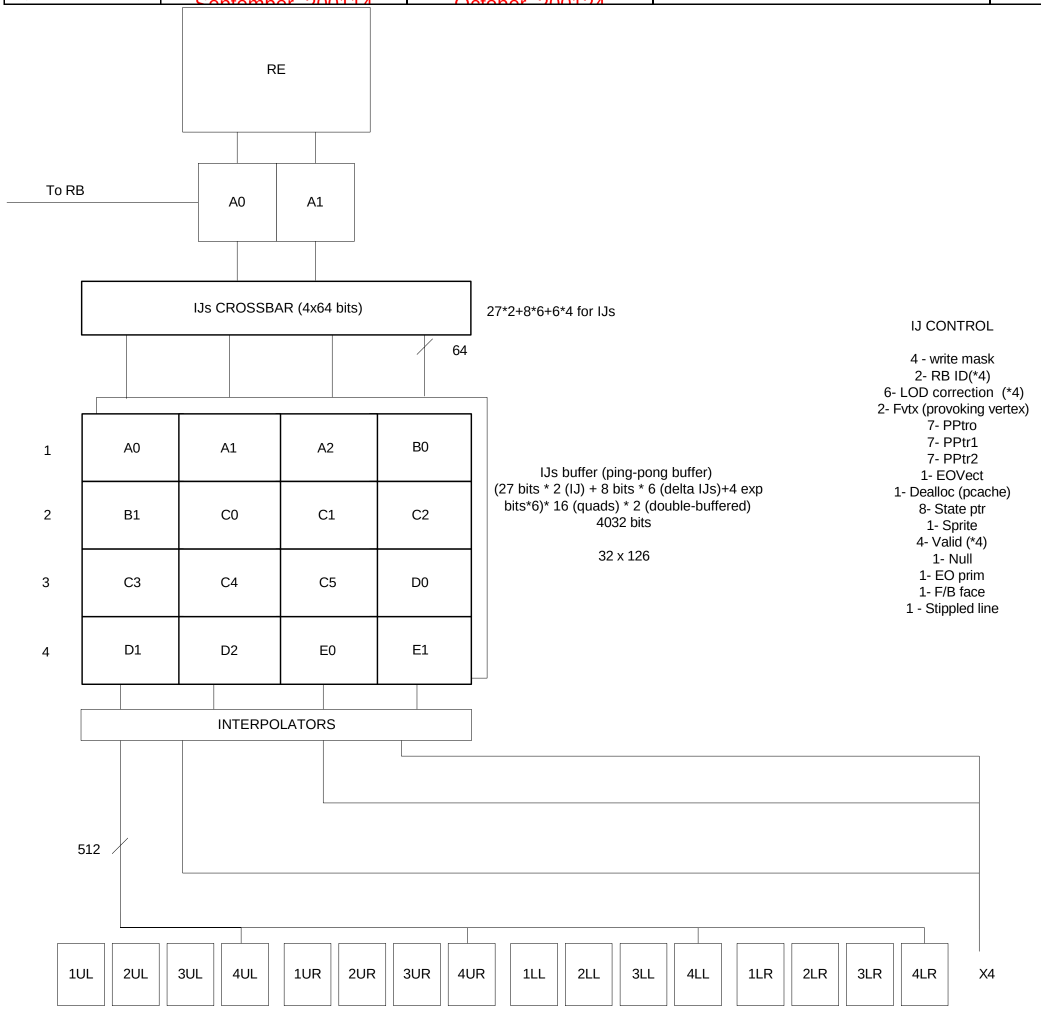
1.3 Control Graph



In green is represented the Texture control interface, in red the ALU control interface, in blue the Interpolated/Vector control interface and in purple is the output file control interface.

2. Interpolated data bus

The interpolators contain an IJ buffer to pack the information as much as possible before writing it to the register file.



RE
To RB
A0
A1
IJs CROSSBAR (4x64 bits)
27*2+8*6+6*4 for IJs
64
1
2
3
4
A0
A1
A2
B0
B1
C0
C1
C2
C3
C4
C5
D0
D1
D2
E0
E1
IJs buffer (ping-pong buffer)
(27 bits * 2 (IJ) + 8 bits * 6 (delta IJs)+4 exp bits*6)* 16 (quads) * 2 (double-buffered)
4032 bits
32 x 126
IJ CONTROL
4 - write mask
2- RB ID(*4)
6- LOD correction (*4)
2- Fvtx (provoking vertex)
7- PPtro
7- PPtr1
7- PPtr2
1- EOVect
1- Dealloc (pcache)
8- State ptr
1- Sprite
4- Valid (*4)
1- Null
1- EO prim
1- F/B face
1 - Stippled line
INTERPOLATORS
512
1UL
2UL
3UL
4UL
1UR
2UR
3UR
4UR
1LL
2LL
3LL
4LL
1LR
2LR
3LR
4LR
X4

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19
SP0	A0	B1	C3	D1					A0	B1	C3	D1					V 0-3	V 16-19	V 32-35	V 48-51
SP1	A1		C4	D2		C0			A1		C4	D2		C0			V 4-7	V 20-23	V 36-39	V 52-55
SP2	A2		C5			C1		E0	A2		C5			C1		E0	V 8-11	V 24-27	V 40-43	V 56-59
SP3					B0	C2	D0	E1					B0	C2	D0	E1	V 12-15	V 28-31	V 44-47	V 60-63
				P0								P1								

Above is an example of a tile we might receive. The IJ information is packed in the IJ buffer 2 quads at a time. The sequencer allows at any given time as many as four quads to interpolate a parameter. They all have to come from the same primitive. Then the sequencer controls the write mask to the register to write the valid data in.~~Since each of the register file is actually physically divided (one 128x128 per MAC) and we don't have the place to hold a maximum size vector of vertices in the parameter buffer, we need to interpolate on a parameter basis rather than on a quad basis. So the order to the register file will be:~~

~~Q0P0 Q1P0 Q2P0 Q3P0 Q0P1 Q1P1 Q2P1 Q3P1 Q0P2 Q1P2 …~~

3. Instruction Store

There is going to be only one instruction store for the whole chip. It ~~may~~ will contain ~~up to 200~~4096~~0~~ instructions of 96 bits each. There is also going to be a control instruction store of size 256(512?)x32.

{ISSUE : The instruction store is loaded by the sequencer using the memory hub ?}.

The read bandwith from this store is ~~24~~ 96*2 bits/ 4 clocks (48 bits/clock)~~/pipe~~. ~~To achieve this this instruction store is likely to be broken up into 4 blocks. An ALU instruction section (1R/1W) split in two and a texture section (1R/1W) also split in two. The bandwith out of those memories is 96 bits/clock.~~It is likely to be a 1R/1W port memory; we use 2 clocks to load the ALU instruction and 2 clocks to load the Texture instruction.

Formatted: Bullets and Numbering

4. Sequencer Instructions

All control flow instructions and move instructions are handled by the sequencer only. The ALUs will perform NOPs during this time (MOV PV,PV, PS,PS).

Formatted: Bullets and Numbering

~~4.~~5. Constant Store

The constant store is managed by the CP. The sequencer is aware of where the constants are using a remaping table also managed by the CP. A likely size for the constant store is 512x128 bits. The constant store is also planned to be shared. The read BW from the constant store is 128~~512/4~~ bits/clock~~/pipe~~ and the write bandwith is 32/4 bits/clock.

In order to do constant store indexing, the sequencer must be loaded first with the indexes (that come from the GPRs). There are 144 wires from the exit of the SP to the sequencer (9 bits pointers x 16 vertexes/clock). Since the data must pass thru the Shader pipe for the float to fixed convertion, there is a latency of 4 clocks (1 instruction) between the time the sequencer is loaded and the time one can index into the constant store. The assembly will look like this

```
MOVA  R1.X,R2.X        // Loads the sequencer with the content of R2.X, also copies the content of R2.X into R1.X
NOP                    // latency of the float to fixed conversion
ADD   R3,R4,C0[R2.X]// Uses the state from the sequencer to add R4 to C0[R2.X] into R3
```

Note that we don't really care about what is in the brackets because we use the state from the MOVA instruction. R2.X is just written again for the sake of simplicity.

The storage needed in the sequencer in order to support this feature is 2*64*9 bits = 1152 bits.

Formatted: Bullets and Numbering

~~5.~~6. Looping and Branches

Loops and branches are planned to be supported and will have to be dealt with at the sequencer level. We plan on supporting constant loops and branches using a control program. The control program has 4(5) instructions:

Formatted: Bullets and Numbering

6.1 The controlling state.

As per Dx9 the following state is available for control flow:

Boolean[15:0]

loop_count[7:0][7:0]
In addition:
loop_start [7:0] [7:0]
loop_step [7:0] [7:0]
Exist to give more control to the controlling program.

We will extend that in the R400 to:
Boolean[31:0]
Loop_count[7:0][15:0]
Loop_Start[7:0] [15:0]
Loop_End[7:0] [15:0]

{ISSUE: How is the controlling state loaded and how many contexts do we have?}

Formatted: Bullets and Numbering

6.2 The Control Flow Program

The R300 uses a match method for control flow: The shader is executed, and at every instruction its address is compared with addresses (or address?) in a control table. The "event" in the control table can redirect operations in the program.

The Method chosen for the R400 is a "control program". The control program has four basic instructions:

Execute
Conditional_execute (Conditional Execute Predicates)
Loop_start
Loop_end

Execute, causes the specified number of instructions in instruction store to be executed.
Conditional_execute checks a condition first, and if true, causes the specified number of instructions in instruction store to be executed.
Loop_start resets the corresponding loop counter to the start value.
Loop_end increments (decrements?) the loop counter and jumps back the specified number of instructions if the loop end condition is not met.

if we try and fit the control flow instructions into 32 bit words, the following instructions are possible choices:

Execute																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	instruction_count												Reserved			Address												

Execute up to 4K instructions at the specified address in the instruction memory.

Conditional Execute																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	Boolean						0: = 0 1: = 1 2,3:NA			Instruction_count						Address												

if the specified boolean (6 bits can address 64 booleans) meets the specified condition then execute the specified instructions (up to 64 instructions)

Conditional Execute Predicates																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	Reserved =0						0: = 0 1: = 1 2,3:NA			Instruction_count						Address												

Check the OR of all current predicate bits. If OR matches the condition execute the specified number of instructions.

Loop_Start

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0																								Loop ID				

Initialize the specified loop

Loop_End

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	Reserved (must = 0)												Start Address								Reserved (must = 0)			Loop ID				

If the loop condition of the current loop is not met, then branch back to the specified address in the control flow program. Note that jumping back to the loop_start results in an infinite loop, the jump should be to loop_start+1.

The way this is described does not prevent nested loops, and the inclusion of the loop id make this easy to do.

The basic model is as follows:

The render state defined the clause boundaries:
Vertex_shader_fetch[7:0][7:0] // eight 8 bit pointers to the location where each clauses control program is located
Vertex_shader_alu[7:0][7:0] // eight 8 bit pointers to the location where each clauses control program is located
Pixel_shader_fetch[7:0][7:0] // eight 8 bit pointers to the location where each clauses control program is located
Pixel_shader_alu[7:0][7:0] // eight 8 bit pointers to the location where each clauses control program is located

The control program can be up to 256 instructions in size. (There is an offset added to the address from the render state before accessing the control flow program memory to allow for multiple programs resident at the same time)

The control program for a given clause is executed to completion before moving to another clause, (with the exception of the pick two nature of the alu execution). The control program is the only program aware of the clause boundaries.

The addresses from the control program are added to another offset to allow for multiple programs resident at the same time.

Under this model, all subroutine calls must be inlined into the control program.

6.3 Data dependant predicate instructions

Formatted: Bullets and Numbering

Data dependant conditionals will be supported in the R400. The only way we plan to support those is by supporting three vector/scalar predicate operations of the form:

PRED_SETE - similar to SETE except that the result is 'exported' to the sequencer.
PRED_SETGT - similar to SETGT except that the result is 'exported' to the sequencer
PRED_SETGTE - similar to SETGTE except that the result is 'exported' to the sequencer

For the scalar operations only we will also support the two following instructions:
PRED_SETE0 – SETE0
PRED_SETE1 – SETE1

The export is a single bit - 1 or 0 that is sent using the same data path as the MOVA instruction. The sequencer will maintain the 64 bit predicate vector and use it to control the write masking (two sets for interleaved operation). This predicate is not maintained across clause boundaries.

Then we have two conditional execute bits. The first bit is a conditional execute "on" bit and the second bit tells us if we execute on 1 or 0. For exemple, the instruction :

P0_ADD R0,R1,R2

Is only going to write the result of the ADD into those GPRs whose predicate bit is 0. Alternatively, P1_ADD would only write the results to the GPRs whose predicate bit is set. The use of the P0 or P1 without precharging the sequencer with a PRED instruction is undefined.

{Issue: do we have to have a NOP between PRED and the first instruction that uses a predicate?}

Formatted: Bullets and Numbering

6.4 Register file indexing

Because we can have loops in texture clause, we need to be able to index into the register file in order to retrieve the data created in a texture clause loop and use it into an ALU clause. The instruction will include the base address for register indexing and the instruction will contain these controls :

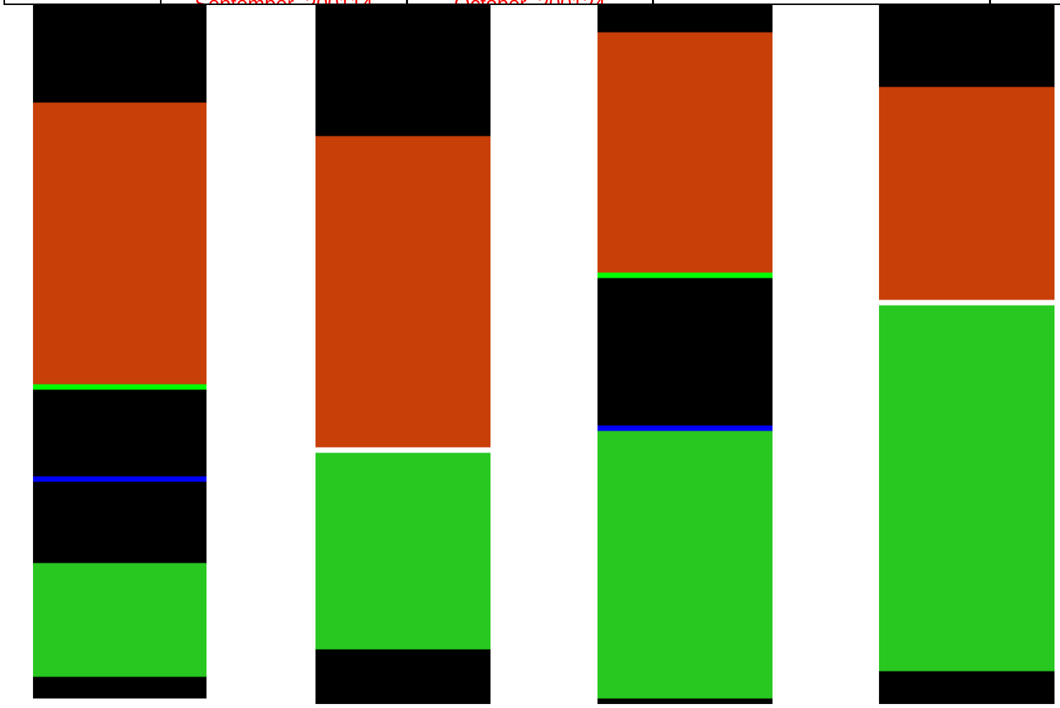
Bit7	Bit 6	
0	0	'absolute register'
0	1	'relative register'
1	0	'previous vector'
1	1	'previous scalar'

In the case of an absolute register we just take the address as is. In the case of a relative register read we take the base address and we add to it the loop counter and this becomes our new address that we give to the shader pipe.~~However, it is still unclear if we plan on supporting data dependent branches or not.~~

Formatted: Bullets and Numbering

~~6.~~7. Register file allocation

The register file allocation for vertices and pixels can either be static or dynamic. In both cases, the register file in managed using two round robins (one for pixels and one for vertices). In the dynamic case the boundary between pixels and vertices is allowed to move, in the static case it is fixed to VERTEX_REG_SIZE for vertices and 256-VERTEX_REG_SIZE for pixels.



Above is an example of how the algorithm works. Vertices come in from top to bottom; pixels come in from bottom to top. Vertices are in orange and pixels in green. The blue line is the tail of the vertices and the green line is the tail of the pixels. Thus anything between the two lines is shared. When pixels meets vertices the line turns white and the boundary is static until both vertices and pixels share the same "unallocated bubble". Then the boundary ~~as~~ is allowed to movi~~nge~~ again.

Formatted: Bullets and Numbering

~~7.~~8. Texture Arbitration

The texture arbitration logic chooses one of the 8 potentially pending texture clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute. Once chosen, the clause state machine will send one 2x2 texture fetch per clock (or 4 fetches in one clock every 4 clocks) until all the texture fetch instructions of the clause are sent. This means that there cannot be any dependencies between two texture fetches of the same clause.

The arbitrator will not wait for the texture fetches to return prior to selecting another clause for execution. The texture pipe will be able to handle up to X(?) in flight texture fetches and thus there can be a fair number of active clauses waiting for their texture return data.

Formatted: Bullets and Numbering

~~8.~~9. ALU Arbitration

ALU arbitration proceeds in almost the same way than texture arbitration. The ALU arbitration logic chooses one of the 8 potentially pending ALU clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute. There are two ALU arbitrers, one for the even clocks and one for the odd clocks. For exemple, here is the sequencing of two interleaved ALU clauses (E and O stands for Even and Odd):

Einst0 Oinst0 Einst1 Oinst1 Einst2 Oinst2 Einst0 Oinst3 Einst1 Oinst4 Einst2 Oinst0…
Proceeding this way hides the latency of 8 clocks of the ALUs.

~~9.~~10. Handling Stalls

Formatted: Bullets and Numbering

When the output file is full, the sequencer prevents the ALU arbitration logic to select the last clause (this way nothing can exit the shader pipe until there is place in the output file. If the packet is a vertex packet and the position buffer is full (POS_FULL) then the sequencer also prevents a thread to enter the exporting clause (4?). The sequencer will set the OUT_FILE_FULL signal n clocks before the output file is actually full and thus the ALU arbitrer will be able read this signal and act accordingly by not preventing exporting clauses to proceed.

~~10.~~11. Content of the reservation station FIFOs

Formatted: Bullets and Numbering

3 bits of Render State 6-7 bits for the base address of the instruction store, ~~and~~ some bits for LOD correction and coverage mask information in order to fetch texture for only valid pixels. Every other information (such as the coverage mask, quad address, etc.) is put in a FIFO and is retrieved when the quad exits the shader pipe to enter in the output file buffer. Since pixels and vertices are kept in order in the shader pipe, we only need two fifos (one for vertices and one for pixels) deep enough to cover the shader pipe latency. This size will be determined later when we will know the size of the small fifos between the reservation stations.

~~11.~~12. The Output File

Formatted: Bullets and Numbering

The output file is where pixels are put before they go to the RBs. The write BW to this store is 256 bits/clock. Just before this output file are staging registers with write BW 512 bits/clock and read BW 256 bits/clock. For this reason only ONE concurrent program can be of clause 8 (exporting clause) the other program MUST not. The staging registers are 4x128 (and there are 16 of those on the whole chip).

13. IJ Format

Formatted: Bullets and Numbering

The IJ information sent by the PA is of this format on a per quad basis:

We have a vector of IJ's (one IJ per pixel at the centroid of the fragment or at the center of the pixel depending on the mode bit). The interpolation is done at a different precision across the 2x2. The upper left pixel's parameters are always interpolated at full 24x24 mantissa precision. Then the result of the interpolation along with the difference in IJ in reduced precision is used to interpolate the parameter for the other three pixels of the 2x2. Here is how we do it:

Assuming P0 is the interpolated parameter at Pixel 0 having the barycentric coordinates I(0), J(0) and so on for P1,P2 and P3. Also assuming that A is the parameter value at V0 (interpolated with I), B is the parameter value at V1 (interpolated with J) and C is the parameter value at V2 (interpolated with (1-I-J).

$$\begin{aligned}
\Delta 01I &= I(1) - I(0) \\
\Delta 01J &= J(1) - J(0) \\
\Delta 02I &= I(2) - I(0) \\
\Delta 02J &= J(2) - J(0) \\
\Delta 03I &= I(3) - I(0) \\
\Delta 03J &= J(3) - J(0)
\end{aligned}$$

P0	P1
P2	P3

$$\begin{aligned}
P0 &= C + I(0)*(A - C) + J(0)*(B - C) \\
P1 &= P0 + \Delta 01I*(A - C) + \Delta 01J*(B - C) \\
P2 &= P0 + \Delta 02I*(A - C) + \Delta 02J*(B - C) \\
P3 &= P0 + \Delta 03I*(A - C) + \Delta 03J*(B - C)
\end{aligned}$$

P0 is computed at full 24x24 mantissa precision and P1 to P3 are computed at 8X24 mantissa precision. So far no visual degradation of the image was seen using this scheme.

Multiplies (Full Precision): 2
Multiplies (Reduced precision): 6
Subtracts 24x24 yielding 8 bits (IJs): 6
Subtracts 24x24 (Parameters): 2
Adds: 8

FORMAT OF P0's IJ : Mantissa 23 Exp 4 for I
Mantissa 23 Exp 4 for J

FORMAT of Deltas (x3): Mantissa 8 Exp 4 for I
Mantissa 8 Exp 4 for J

Total number of bits : 23*2 + 8*6 + 4*8 = 126 (rounded up on the bus to 128)

Formatted: Bullets and Numbering

14. The parameter cache

The parameter cache is where the vertex shaders export their data. It consists of 16 128x128 memories (1R/1W). The reuse engine will make it so that all vertexes of a given primitive will hit different memories.~~(RB FIFO and Parameter Cache)~~

Formatted: Bullets and Numbering

15. ~~The output file is where program results are exported when the pixel/vertex shader finishes. It constists of a 512x128 memory cell that is statically divided between pixels and vertices. The output file has 1 write port and 1 read port. The sequencer is responsible for managing the addresses of this output file and for stalling the shader pipe should this output file fill up. The management is done by keeping the tail and head pointers of each sections (pixels and vertices) and incrementing them using a simple RoundRobin allocation policy. The sequencer must also arbitrate between the PA and the RB for the use of the read port. This arbitration will either be priority based or just interleaved evenly (1 read every 2 clocks for each of the blocks).~~Vertex position exporting

On clause 4 (or 5) the vertex shader can export to the PA both the vertex position and the point sprite. It can also do so at clause 8 if not done at clause 4. The export is done by putting the exported position back into the GPRs. Then using the texture port in an opportunistic manner, 16 positions are put into a FIFO (16x128) in order (left to right). This fifo drains 128 bits per clock to the PA and once empty is filled up again with sprite sizes (if any). The process is repeated 4 times. The sequencer must make sure that the program doesn't enter ALU clause 5 (it can enter texture clause 5) because the registers can be reused at this point. The sequencer must also make sure not to dealocate an exporting program before it is done exporting data. Along with the position is exported a pointer to the parameter cache where the data will be once the vertex shader exports.

Formatted: Bullets and Numbering

16. Registers

DYNAMIC_REG	Dynamic allocation (pixel/vertex) of the register file on or off.
VERTEX_REG_SIZE	What portion of the register file is reserved for vertices (static allocation only)
PIXEL_MIN_SIZE	Minimal size of the register file's pixel portion (dynamic only)
VERTEX_MIN_SIZE	Minimal size of the register file's vertex portion (dynamic only)
Vshader_fetch[7:0][7:0]	eight 8 bit pointers to the location where each clauses control program is located
Vshader_alu[7:0][7:0]	eight 8 bit pointers to the location where each clauses control program is located
Pshader_fetch[7:0][7:0]	eight 8 bit pointers to the location where each clauses control program is located

Pshader_alu[7:0][7:0]	eight 8 bit pointers to the location where each clauses control program is located
PSHADER	base pointer for the pixel shader
VSHADER	base pointer for the vertex shader
PCNTLSHADER	base pointer for the pixel control program
VCNTLSHADER	base pointer for the vertex control program
VWRAP	wrap point for the vertex shader instruction store
PWRAP	wrap point for the pixel shader instruction store
REG_ALLOC_PIX	number of registers to allocate for pixel shader programs
REG_ALLOC_VERT	number of registers to allocate for vertex shader programs
PARAM_MASK[0…16]	parameter mask to specify wich parameters the pixel shader
FLAT_GOUR[0…16]	wich parameters are to be gouraud shaded
GEN_TEX[0….16]	for wich parameters do we need to generate tex coords.
CYL_WRAP[0…64]	for wich vertices do we do the cyl wrapping.
P_EXPORT	number of exports for pixel shader
V_EXPORT	number of exports for vertex shader (also the number of interpolated parameters for pixel shaders)
V_EXPORT_LOC	Vertex shader exporting to RB or the PCACHE

Formatted: Bullets and Numbering

~~12.~~17. Interfaces

~~12.1~~17.1 External Interfaces

~~12.1.1~~17.1.1 *~~Sequencer to Shader Engine Bus~~PA/SC to RE : IJ bus*

~~This is a bus that sends the instruction and constant data to all 4 Sub-Engines of the Shader. Because a new instruction is needed only every 4 clocks, the width of the bus is divided by 4 and both constants and instruction are sent over those 4 clocks.~~ This is a bus that sends the IJ information to the IJ fifos on the top of each shader pipe. At the same time the control information goes to the sequencer

Name	Direction	Bits	Description
~~Instruction Start~~IJs	~~SEQ->SP~~PA→RE	~~1~~64	~~High on first cycle of transfer~~IJ information sent over 2 clocks

Formatted: Bullets and Numbering

~~12.1.2~~

~~12.1.2Shader Engine to Output File~~

~~Every clock each Sub-Engine can output 128 bits of 'vector' data and 32 bits of 'scalar' data to an output file (?). This data will be compressed into 128 bits total prior to storage in output file.~~

Formatted: Bullets and Numbering

17.1.2 *PA/SC to SEQ : IJ Control bus*

This is the control information sent to the sequencer in order to control the IJ fifos and all other information needed to execute a shader program on the sent pixels.

Name	Direction	Bits	Description
Write Mask	PA→SEQ(RE)	4	Quad Write mask left to right
RB_ID	PA→SEQ(RE)	8	RB id for each quad sent 2 bits per quad
LOD_CORRECT	PA→SEQ(RE)	24	LOD correction per quad (6 bits per quad)
FVTX	PA→SEQ(RE)	2	Provoking vertex for flat shading
PPTR0	PA→SEQ(RE)	11	P Store pointer for vertex 0
PPRT1	PA→SEQ(RE)	11	P Store pointer for vertex 1
PPTR2	PA→SEQ(RE)	11	P Store pointer for vertex 2
E_OFF_VECTOR	PA→SEQ(RE)	1	End of the vector

DEALLOC	PA→SEQ(RE)	1	Deallocation token for the P Store
STATE	PA→SEQ(RE)	21	State/constant pointer (6*3+3)
SPRITE	PA→SEQ(RE)	1	Need to generate tex cords
VALID	PA→SEQ(RE)	16	Valid bits for all pixels
NULL	PA→SEQ(RE)	1	Null Primitive (for PC deallocation purposes)
E_OFF_PRIM	PA→SEQ(RE)	1	End Of the primitive
FBFACE	PA→SEQ(RE)	1	Front face = 1, back face = 0
STIPPLE_LINE	PA→SEQ(RE)	1	Stippled line need to load tex cords from alternate buffer
RTRn	SEQ→PA	1	Stalls the PA in n clocks
RTS	PA→SEQ(RE)	1	PA ready to send data
QuadX	PA→SEQ(RE)	40	Quad X address 10 bits per quad
QuadY	PA→SEQ(RE)	40	Quad Y address 10 bits per quad

17.1.3 *PA/SC to RE : Vertex Bus*

Formatted: Bullets and Numbering

Name	**Direction**	**Bits**	**Description**
Vertex indexes	PA→RE	32	Pointers of indexes

17.1.4 *PA/SC to SEQ : Vertex Control Bus*

Formatted: Bullets and Numbering

Name	**Direction**	**Bits**	**Description**
STATE	PA→SEQ	21	Render State (6*3+3 for constants)
Position Cache Pointer	PA→SEQ	7	Pointer to the position cache
Write Mask	PA→SEQ	64?	Which vertices are valid
E_OFF_VECTOR	PA→SEQ	1	End of the vector

17.1.5 *CP to SEQ : Constant store load*

Formatted: Bullets and Numbering

Name	**Direction**	**Bits**	**Description**
Constant Address	CP→SEQ	8	Address of the block of 4 constants
Constant Data	CP→SEQ	512	Data sent over 4 clocks
Remap Address	CP→SEQ	10	Remaping address write address
Remap Data pointer	CP→SEQ	8	Remaping pointer

17.1.6 *CP to SEQ : Texture State store load*

Formatted: Bullets and Numbering

Name	**Direction**	**Bits**	**Description**
Constant Address	CP→SEQ	8	Address of the block of 4 state constants
Constant Data	CP→SEQ	512	Data sent over 4 clocks
Remap Address	CP→SEQ	10	Remaping address write address
Remap Data pointer	CP→SEQ	8	Remaping pointer

17.1.7 *CP to SEQ : Control State store load*

Formatted: Bullets and Numbering

Name	**Direction**	**Bits**	**Description**

{ISSUE: How,Who and what is the size of this bus?}

17.1.8 *MH to SEQ: Instruction store Load*

Formatted: Bullets and Numbering

Name	**Direction**	**Bits**	**Description**
Instruction address	MH→SEQ	12	Instruction address
Instruction	MH→SEQ	96	Instruction X times
Control Instruction address	MH→SEQ	9	Pointer to the control instruction store
Control Instruction	MH→SEQ	32	Control Instruction X times

17.1.9 SP to RB : Pixel read from RBs

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
Pixel Data	SP→RB	256	2 pixels (or ½ quad)
Quad Address	SP→RB	20	XY address 10 bits per

Only one exporting clause (7) can be selected at any given time.

17.1.10 SP to PA/SC : Position return bus

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
Position return	SP→PA	128	Position data or sprite size
Position Buffer pointer	SP→PA	7	Pointer to the position cache
Parameter cache pointer	SP→PA	11	Pointer where the data will be in the parameter cache

For point sprites and position exports the size and position are interleaved on a 16 x 16 basis. We export 16 positions then 16 point sprite sizes. The registers are taken until the next ALU clause where they are going to be available again. Thus the sequencer has to make sure that we finished exporting data before allowing the program in the next ALU clause.

~~12.1.3~~17.1.11 Shader Engine to Texture Unit Bus (Fast Bus)

Formatted: Bullets and Numbering

~~One~~ Four quad's worth of addresses is transferred to Texture Unit every clock. These are sourced from a different pixel within each of the sub-engines repeating every 4 clocks. The register file index to read must precede the data by 2 clocks. The Read address associated with Quad 0 must be sent 1 clock after the Instruction Start signal is sent, so that data is read 3 clocks after the Instruction Start.

~~One~~Four Quad's worth of Texture Data may be written to the Register file every clock. These are directed to a different pixel of the sub-engines repeating every 4 clocks. The register file index to write must accompany the data. Data and Index associated with the Quad 0 must be sent 3 clocks after the Instruction Start signal is sent.

Name	Direction	Bits	Description
Tex_Read_Register_Index	SEQ->SP	~~8~~7	Index into Register files for reading Texture Address
Tex_RegFile_Read_Data	SP->TEX	~~512~~2048	~~4~~ 16 Texture Addresses read from the Register file
Tex_Write_Register_Index	SEQ->TEX	~~8~~7	Index into Register file for write of returned Texture Data

~~12.1.4~~17.1.12 Sequencer to Texture Unit bus (Slow Bus)

Formatted: Bullets and Numbering

Once every four clock, the texture unit sends to the sequencer on wich clause it is now working and if the data in the registers is ready or not. This way the sequencer can update the texture counters for the reservation station fifos. The sequencer also provides the intruction and constants for the texture fetch to execute and the address in the register file where to write the texture return data.

Name	Direction	Bits	Description
Tex_Ready	TEX→ SEQ	1	Data ready
Tex_Clause_Num	TEX→ SEQ	3	Clause number
Tex_cst	SEQ→TEX	~~?~~10	Texture ~~constants X~~state address 10 bits sent over 4 clocks
Tex_Inst	SEQ→TEX	~~?~~12	Texture fetch instruction ~~X~~address 12 bits sent over 4 clocks
EO_CLAUSE	SEQ→TEX	1	Last instruction of the clause
PHASE	SEQ→TEX	1	Write phase signal

Formatted: Bullets and Numbering

18. Internal interfaces

~~12.1.5 *Shader Engine to RE/PA Bus*~~

~~12.1.6 *PA? to sequencer*~~

~~13.~~19. Examples of program executions

~~13.1.1~~19.1.1 *Sequencer Control of a Vector of Vertices*

1. PA sends a vector of ~~16~~ 64 vertices (actually vertex indices – 32 bits/index for ~~512~~ 2048 bit total) to the RE's Vertex FIFO
   - state pointer as well as tag into position cache is sent along with vertices
   - space was allocated in the position cache for transformed position before the vector was sent
   - **also before the vector is sent to the RE, the CP has loaded the global instruction store with the vertex shader program (using the MH?)**
   - The vertex program is assumed to be loaded when we receive the vertex vector.
     - the SEQ then accesses the IS base for this shader using the local state pointer (provided to all sequencers by the RBBM when the CP is done loading the program)
2. SEQ arbitrates between the Pixel FIFO and the Vertex FIFO – basically the Vertex FIFO always has priority
   - at this point the vector is removed from the Vertex FIFO
   - the arbitrer is not going to select a vector to be transformed if the parameter cache is full unless the pipe as nothing else to do (ie no pixels are in the pixel fifo).
3. SEQ allocates space in the SP register file for index data plus GPRs used by the program
   - the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer that came down with the vertices
   - SEQ will not send vertex data until space in the register file has been allocated
4. SEQ sends the vector to the SP register file over the RE_SP interface (which has a bandwidth of ~~512~~ 2048 bits/cycle)
   - the ~~16~~ 64 vertex indices are sent to the ~~16~~ 64 register files over 4 cycles
     - RF0 of SU0, SU1, SU2, and SU3 is written the first cycle
     - RF1 of SU0, SU1, SU2, and SU3 is written the second cycle
     - RF2 of SU0, SU1, SU2, and SU3 is written the third cycle
     - RF3 of SU0, SU1, SU2, and SU3 is written the fourth cycle
   - the index is written to the least significant 32 bits **(floating point format?) (what about compound indices)** of the 128-bit location within the register file (w); the remaining data bits are set to zero (x, y, z)
5. SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of texture state machine 0, or TSM0 FIFO)
   - the control packet contains the state pointer, the tag to the position cache and a register file base pointer.
6. TSM0 accepts the control packet and fetches the instructions for texture clause 0 from the global instruction store
   - TSM0 was first selected by the TSM arbiter before it could start
7. all instructions of texture clause 0 are issued by TSM0
8. the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)
   - TSM0 does not wait for requests made to the Texture Unit to complete; it passes the register file write index for the texture data to the TU, which will write the data to the RF as it is received
   - once the TU has written all the data to the register files, it increments a counter that is associated with ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go ahead start to execute the ALU clause

9. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store

10. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of texture state machine 1, or TSM1 FIFO)

11. the control packet continues to travel down the path of reservation stations until all clauses have been executed
    - position can be exported in ALU clause 3 (or 4?); the data (and the tag) is sent over a position bus (which is shared with all four shader pipes) back to the PA's position cache
    - A parameter cache pointer is also sent along with the position data. This tells to the PA where the data is going to be in the parameter cache.
        - there is a position export FIFO in the SP that buffers position data before it gets sent back to the PA
        - the ASM arbiter will prevent a packet from starting an exporting clause if the position export FIFO is full
    - parameter data is exported in clause 7 (as well as position data if it was not exported earlier)
        - parameter data is sent to the Parameter Cache over a dedicated bus
        - the SEQ allocates storage in the Parameter Cache, and the SEQ deallocates that space when there is no longer a need for the parameters (it is told by the PA when using a token).
        - the ASM arbiter will prevent a packet from starting on ASM7 if the parameter cache (or the position buffer if position is being exported) is full

12. after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program

~~13.1.2~~19.1.2 *Sequencer Control of a Vector of Pixels*

Formatted: Bullets and Numbering

1. **As with vertex shader programs, pixel shaders are loaded into the global instruction store by the CP**
    - At this point it is assumed that the pixel program is loaded into the instruction store and thus ready to be read.

~~2.the RE's Parameter Buffer is loaded from the Parameter Cache before the SEQ takes control of the vector~~

Formatted: Bullets and Numbering

- ~~after the HZ culling stage a request is made by the RE to send parameter data to the Parameter buffer~~
- ~~the Parameter buffer is wide enough to source 3 vertices worth of a particular parameter in one cycle~~
- ~~**at this moment the right sequencer will free up the parameter store locations not used anymore using the token provided by the PA**.~~

~~3.~~2. the RE's Pixel FIFO is loaded with the barycentric coordinates for pixel quads by the detailed walker
- the state pointer and the LOD correction bits are also placed in the Pixel FIF0
- the Pixel FIFO is wide enough to source ~~one~~ four quad's worth of barycentrics per cycle

~~4.~~3. SEQ arbitrates between Pixel FIFO and Vertex FIFO – when there are no vertices pending OR there is no space left in the register files for vertices, the Pixel FIFO is selected

Formatted: Bullets and Numbering

~~5.~~4. SEQ allocates space in the SP register file for all the GPRs used by the program
- the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer
- SEQ will not allow interpolated data to be sent to the shader until space in the register file has been allocated

~~6.SEQ control starts with the interpolation of parameters (up to 16 per thread) by sending the barycentric coordinates from the Pixel FIFO and the parameters from the Parameter Buffer to the interpolator~~

Formatted: Bullets and Numbering

- ~~P0i, P0j, and P0k (the value of P0 at each vertex) are loaded into the interpolator from the Parameter buffer~~
- ~~Q0 i, j, and k are loaded into the interpolator from the Pixel FIFO~~
- ~~The interpolator then generates the parameter value for each pixel in Q0 (Q0P0)~~
- ~~**P0i, P0j, and P0k are sent to the interpolator for Q1 only if Q1 is from a different primitive; if Q1 is from the same primitive as Q0, then the P0i, P0j, and P0k values loaded for Q0 are held by the interpolator and reused for Q1**~~
    - ~~**a "different_prim" control bit is passed with the barycentric data for each quad in the Pixel FIFO that indicates whether new parameter data needs to be loaded into the interpolator**~~
- ~~Q1 i, j, and k are then loaded into the interpolator from the Pixel FIFO~~
- ~~The interpolator then generates the parameter value for each pixel in Q1 (Q1P0)~~
- ~~Q2P0 and Q3P0 are generated in a similar manner~~
- ~~The next set of parameter data - P1i, P1j, and P1k - is then loaded into the interpolator~~

- ~~Q0 i, j, and k now must be re-read from the Pixel FIFO – this means that the output of the Pixel FIFO loops through the top four entries on each read command until at the end a final “block_pop” signal is asserted, causing the top four sets of barycentric coordinates to finally be removed~~
- ~~so the order of parameter info generated is Q0P0, Q1P0, Q2P0, Q3P0, Q0P1, Q1P1, etc.~~

~~7.~~5. SEQ controls the transfer of interpolated data to the SP register file over the RE_SP interface (which has a bandwidth of ~~512~~ 2048 bits/cycle). See interpolated data bus diagrams for details.

- ~~16 pixels worth of interpolated parameter data is sent to the 16 register files over 4 cycles~~
    - ~~RF0 of SU0, SU1, SU2, and SU3 is written with Q0P0 the first cycle~~
    - ~~RF1 of SU0, SU1, SU2, and SU3 is written with Q1P0 second cycle~~
    - ~~RF2 of SU0, SU1, SU2, and SU3 is written with Q2P0 third cycle~~
    - ~~RF3 of SU0, SU1, SU2, and SU3 is written with Q3P0 fourth cycle~~

~~8.~~6. SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of texture state machine 0, or TSM0 FIFO)

- note that there is a separate set of reservation stations/arbiters/state machines for vertices and for pixels
- the control packet contains the state pointer, the register file base pointer, and the LOD correction bits
- all other informations (such as quad address for example) travels in a separate FIFO

~~9.~~7. TSM0 accepts the control packet and fetches the instructions for texture clause 0 from the global instruction store

Formatted: Bullets and Numbering

- TSM0 was first selected by the TSM arbiter before it could start

~~10.~~8. all instructions of texture clause 0 are issued by TSM0

Formatted: Bullets and Numbering

~~11.~~9. the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)

- TSM0 does not wait for texture requests made to the Texture Unit to complete; it passes the register file write index for the texture data to the TU, which will write the data to the RF as it is received
- once the TU has written all the data for a particular clause to the register files, it increments a counter that is associated with the ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go ahead and pop the FIFO and start to execute the ALU clause

~~12.~~10. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store

Formatted: Bullets and Numbering

~~13.~~11. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of texture state machine 1, or TSM1 FIFO)

~~14.~~12. the control packet continues to travel down the path of reservation stations until all clauses have been executed

- pixel data is exported in the last ALU clause (clause 7)
    - it is sent to an output FIFO where it will be picked up by the render backend
    - the ASM arbiter will prevent a packet from starting on ASM7 if the output FIFO is full

~~15.~~13. after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program

Formatted: Bullets and Numbering

~~13.1.3~~19.1.3 *Notes*

Formatted: Bullets and Numbering

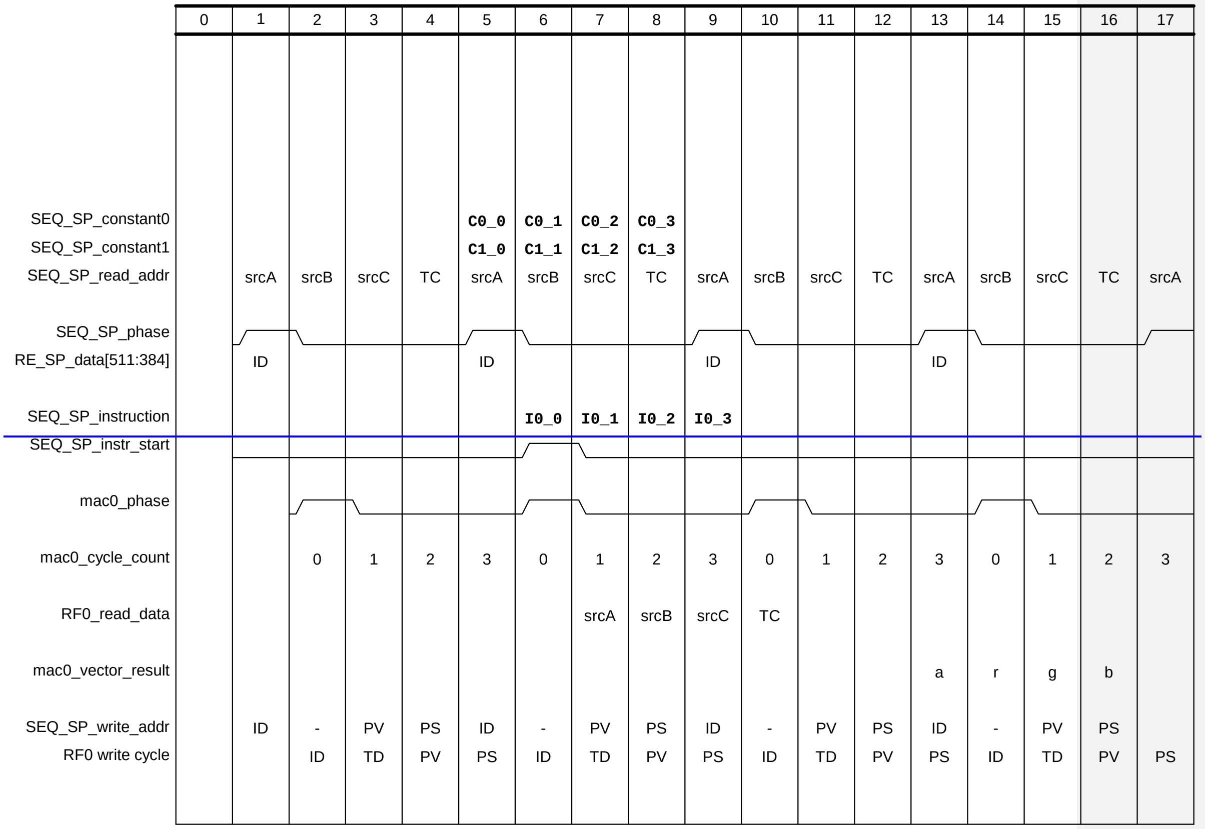
~~16.~~14. the state machines and arbitrers will operate ahead of time so that they will be able to immediately start the real threads or stall.

~~17.~~15. the register file base pointer for a vector needs to travel with the vector through the reservation stations, but the instruction store base pointer does not – this is because the RF pointer is different for all threads, but the IS pointer is only different for each state and thus can be accessed via the state pointer

16. Waterfalling, parameter buffer allocation, loops and branches and parameter cache de-allocation still needs to be specked out.

~~14. Timing Diagrams~~

~~14.1 MAC 0~~

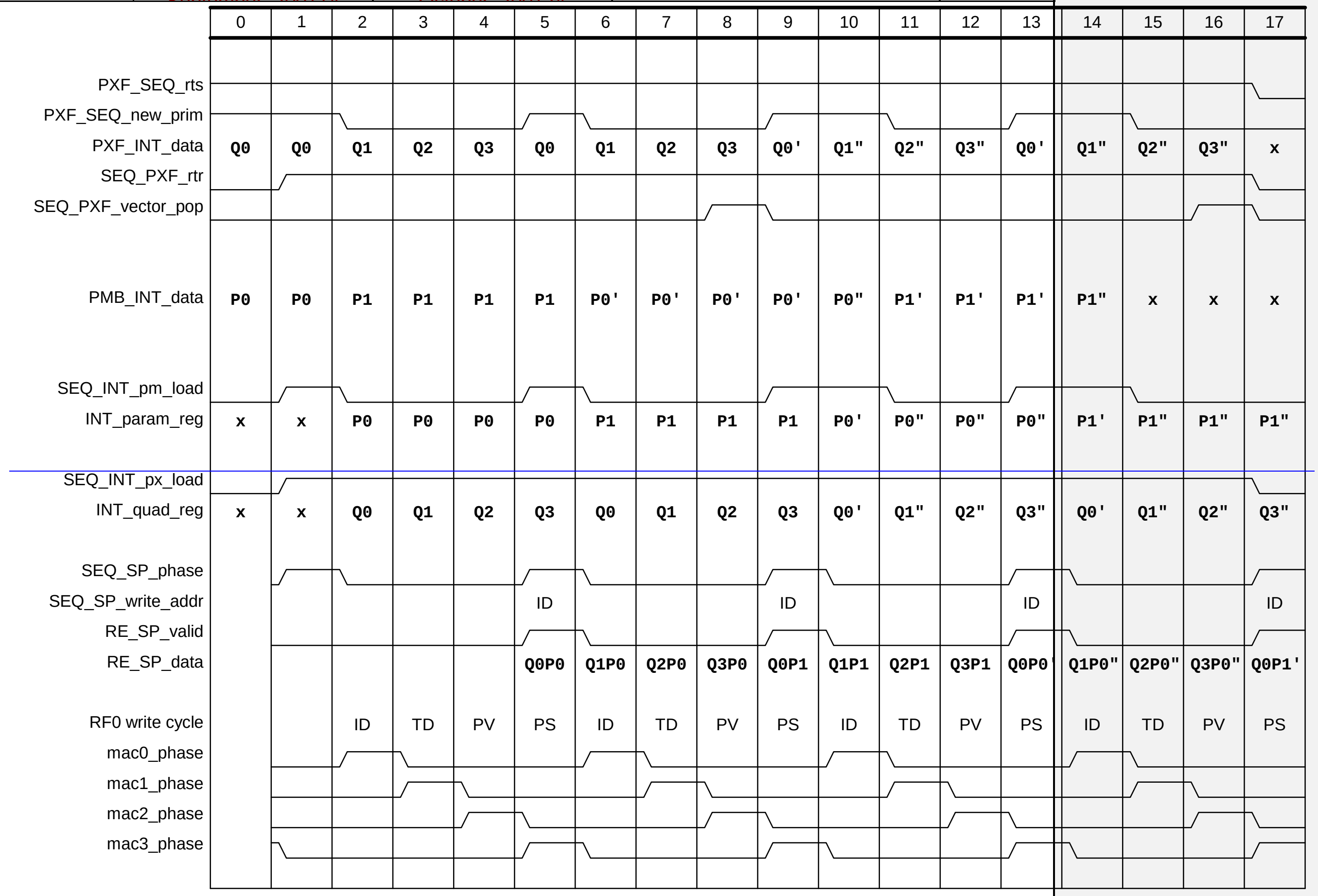


	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
SEQ_SP_constant0						C0_0	C0_1	C0_2	C0_3									
SEQ_SP_constant1						C1_0	C1_1	C1_2	C1_3									
SEQ_SP_read_addr		srcA	srcB	srcC	TC	srcA	srcB	srcC	TC	srcA	srcB	srcC	TC	srcA	srcB	srcC	TC	srcA
SEQ_SP_phase																		
RE_SP_data[511:384]		ID				ID				ID				ID				
SEQ_SP_instruction							I0_0	I0_1	I0_2	I0_3								
SEQ_SP_instr_start																		
mac0_phase																		
mac0_cycle_count			0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
RF0_read_data								srcA	srcB	srcC	TC							
mac0_vector_result														a	r	g	b	
SEQ_SP_write_addr		ID	-	PV	PS	ID	-	PV	PS	ID	-	PV	PS	ID	-	PV	PS	
RF0 write cycle			ID	TD	PV	PS	ID	TD	PV	PS	ID	TD	PV	PS	ID	TD	PV	PS

Timing Diagram 1: Sequencer to Shader Pipe 0, Shader Unit 0, MAC 0

~~14.2 Sequencer to Shader Pipe~~

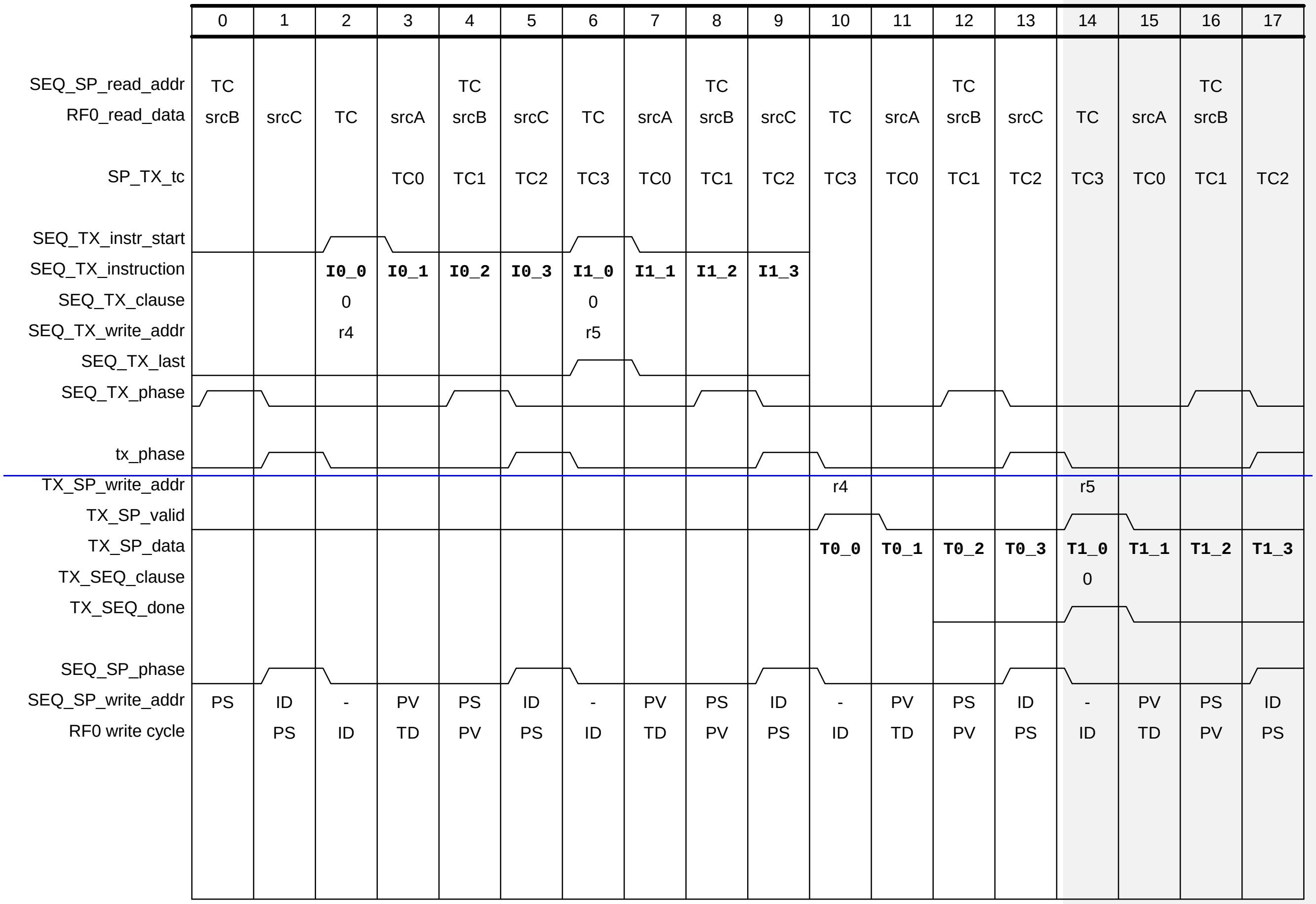
Formatted: Bullets and Numbering


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
PXF_SEQ_rts
PXF_SEQ_new_prim
PXF_INT_data: Q0 Q0 Q1 Q2 Q3 Q0 Q1 Q2 Q3 Q0' Q1" Q2" Q3" Q0' Q1" Q2" Q3" x
SEQ_PXF_rtr
SEQ_PXF_vector_pop
PMB_INT_data: P0 P0 P1 P1 P1 P1 P0' P0' P0' P0' P0" P1' P1' P1' P1" x x x
SEQ_INT_pm_load
INT_param_reg: x x P0 P0 P0 P0 P1 P1 P1 P1 P0' P0" P0" P0" P1' P1" P1" P1"
SEQ_INT_px_load
INT_quad_reg: x x Q0 Q1 Q2 Q3 Q0 Q1 Q2 Q3 Q0' Q1" Q2" Q3" Q0' Q1" Q2" Q3"
SEQ_SP_phase
SEQ_SP_write_addr: ID ID ID ID
RE_SP_valid
RE_SP_data: Q0P0 Q1P0 Q2P0 Q3P0 Q0P1 Q1P1 Q2P1 Q3P1 Q0P0 Q1P0" Q2P0" Q3P0" Q0P1'
RF0 write cycle: ID TD PV PS ID TD PV PS ID TD PV PS ID TD PV PS
mac0_phase
mac1_phase
mac2_phase
mac3_phase

Timing Diagram 2: RE Interpolator to Shader Pipe Data Transfer

Formatted: Bullets and Numbering

~~14.3 Sequencer to Texture Pipe~~


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
SEQ_SP_read_addr TC TC TC TC TC
RF0_read_data srcB srcC TC srcA srcB srcC TC srcA srcB srcC TC srcA srcB srcC TC srcA srcB
SP_TX_tc TC0 TC1 TC2 TC3 TC0 TC1 TC2 TC3 TC0 TC1 TC2 TC3 TC0 TC1 TC2
SEQ_TX_instr_start
SEQ_TX_instruction I0_0 I0_1 I0_2 I0_3 I1_0 I1_1 I1_2 I1_3
SEQ_TX_clause 0 0
SEQ_TX_write_addr r4 r5
SEQ_TX_last
SEQ_TX_phase
tx_phase
TX_SP_write_addr r4 r5
TX_SP_valid
TX_SP_data T0_0 T0_1 T0_2 T0_3 T1_0 T1_1 T1_2 T1_3
TX_SEQ_clause 0
TX_SEQ_done
SEQ_SP_phase
SEQ_SP_write_addr PS ID - PV PS ID - PV PS ID - PV PS ID - PV PS ID
RF0 write cycle ID PS ID TD PV PS ID TD PV PS ID TD PV PS ID TD PV PS

Timing Diagram 3: Sequencer - Texture Unit Interface and Texture Unit - Shader Pipe Data Transfer

~~14.4 Timing diagrams explanations~~

Formatted: Bullets and Numbering

~~The numbering of the four shader pipes, the four shader units, and the four MACs is from left to right and from 0 to 3. So for example the most significant 512 bits of a SP goes to SU0 and the least significant 512 bits go to SU3; within SU0, the most significant 128 bits go to MAC0 and the least significant 128 bits go to MAC3.~~

~~The following assumptions are made:~~

Formatted: Bullets and Numbering

1. ~~all block to block signals are register to register~~
2. ~~for register file reads, the RF read data is available in the MAC one clock after a RF read address is registered into the MAC (this is the same as saying the read data is valid out of the RF two clocks after the address is asserted on the SEQ to SP interface)~~

~~14.4.1 *Timing Diagram 1: Sequencer to Shader Pipe 0, Shader Unit 0, MAC 0*~~

~~This diagram shows the basics of the Sequencer to Shader Pipe interface. For simplicity only the timing relative to MAC0 is shown. The timing for MAC1 is one clock later than MAC0, MAC2 one clock later than MAC1, etc. This means that most of the signals need to be delayed in the SP by one cycle for MAC1, two cycles for MAC2, and three cycles for MAC3.~~

~~**SEQ_SP_constant0**: Constant 0 (128 bits over 4 cycles). Pipelined in SP for other MACs.~~

~~**SEQ_SP_constant1**: Constant 1 (128 bits over 4 cycles). Pipelined in SP for other MACs.~~

~~**SEQ_SP_read_addr**: Register File Read Address (8 bits). Pipelined in SP for other MACs.~~

~~**SEQ_SP_phase**: This signal syncs the data transfer to the RF from the RE, as well as defining the order of all writes into the RF. It is asserted during the cycle that interpolated data (ID) is valid on the RE_SP_ID bus. Pipelined in SP for other MACs.~~

~~**RE_SP_ID[511:384]**: This is the most significant 128 bits of the RE_SP_data interface (meaning that this MAC0 is in SU0).~~

~~**SEQ_SP_instruction**: 96 bits of instruction are sent over 4 cycles. Pipelined in SP for other MACs.~~

~~**SEQ_SP_instr_start**: control bit that signals the first cycle of the instruction transfer. Pipelined in SP for other MACs.~~

~~**mac0_phase**: registered version of SEQ_SP_phase used in MAC0 (this may not be he actual signal name).~~

~~**mac0_cycle_count**: a counter inside the MAC that keeps track of the RF write cycles; 0 here corresponds to the cycle RE interpolated data is written (this may not be he actual signal name).~~

~~**RF0_read_data**: data that is read out of MAC0's register file (this may not be he actual signal name).~~

~~**mac0_vector_result**: the 32-bit output of the vector ALU (PV is built up over 4 cycles) (this may not be he actual signal name).~~

~~**SEQ_SP_write_addr**: Register File Write Address (8 bits). Note that the SEQ does not send the Texture Data write address over this bus. Pipelined in SP for other MACs.~~

~~**RF0 write cycle**: the cycles allocated to the different write sources (ID = Interpolated Data, TD = Texture Data, PV = Previous Vector, PS = Previous Scalar) (not a signal – just a reference point on the diagram).~~

~~14.4.2 *Timing Diagram 2: RE Interpolator to Shader Pipe Data Transfer*~~

Formatted: Bullets and Numbering

~~This diagram shows how pixel data (barycentric coordinates i, j, and k) is sent from the Pixel FIFO to the interpolator under SEQ control, and how parameter data (for each vertex) is also sent to the interpolator under SEQ control. The output of the Interpolator is then shown being sent over the RE_SP interface.~~

~~**PXF_SEQ_rts**: Indicates that the output of the pixel FIFO is valid.~~

~~**PXF_SEQ_new_prim**: The current output of the Pixel FIFO is from a different primitive that the previous output. Tells the SEQ that new parameter info must be fetched (if its not from a new prim, then new parameter data is not needed).~~

~~**PXF_INT_data**: Data output of the Pixel FIFO – goes to the Interpolator.~~

~~**SEQ_PXF_rtr**: Indicates that the current Pixel FIFO output will be taken by the Interpolator (driven by SEQ). Then next quad of data will be driven the next cycle.~~

~~**SEQ_PXF_vector_pop**: SEQ tells the Pixel FIFO to pop a vector of pixels (otherwise RTRs cause the data to be cycled between the four quads).~~

~~**PMB_INT_data**: Data from the Parameter Buffer to the Interpolator. (Note that the control of the parameter buffer is TBD).~~

~~**SEQ_INT_pm_load**: controls the loading of parameter data into the Interpolator.~~

~~**INT_param_reg**: register in the Interpolator that holds the per-vertex parameter data while the per-pixel parameters are generated for one or more quads (may not be the actual signal name).~~
~~**SEQ_INT_px_load**: controls the loading of pixel data into the Interpolator.~~
~~**INT_quad_reg**: : register in the Interpolator that holds one quad's worth of pixel data(may not be the actual signal name).~~

~~**SEQ_SP_phase**: see above under TD1.~~
~~**SEQ_SP_write_addr**: see above under TD1.~~
~~**RE_SP_valid**: Interpolator Data Valid – indicates that the SP should write the ID on the appropriate cycle.~~
~~**RE_SP_data**: Data from the RE interpolator to the SP.~~
~~**RF0 write cycle**: see above under TD1.~~
~~**mac*_phase**: see above under TD1. These phase signals help to show the timing offset between the MACs. Note also that each Shader Unit has a set of these signals (all with the same timing).~~

~~14.4.3 *Timing Diagram 3: Sequencer - Texture Unit Interface and Texture Unit - Shader Pipe Data Transfer*~~

Formatted: Bullets and Numbering

~~This diagram starts with the texture coordinate read from the register file and its transfer to the TX. The instruction transfer is then shown, followed by the texture data transfer to the shader pipe.~~
~~**SEQ_SP_read_addr**: see above. Here shows the cycle that the texture coordinate read address is asserted.~~
~~**RF0_read_addr**: see above.~~
~~**SP_TX_tc**: Texture coordinate data sent from the shader pipe to the texture unit.~~
~~**SEQ_TX_instr_start**: Asserted on the first cycle of a SEQ to TX instruction transfer.~~
~~**SEQ_TX_instruction**: 96 bits of texture instruction transferred over 4 cycles.~~
~~**SEQ_TX_clause**: the clause number associated with this instruction.~~
~~**SEQ_TX_write_addr**: RF write index used by TX for returned texture data.~~
~~**SEQ_TX_last**: indicates that this is the last texture instruction of a clause.~~
~~**SEQ_TX_phase**: syncs the texture data write. Note that it is asserted early enough to be registered into TX and still allow TX to source the texture data to the SP on the correct cycle.~~

~~**tx_phase**: the phase signal after being registered into TX.~~
~~**TX_SP_write_addr**: RF write index for texture data.~~
~~**TX_SP_valid**: indicates that valid texture data is being driven to the SP.~~
~~**TX_SP_data**: the texture data.~~
~~**TX_SEQ_clause**: the clause number associated with the texture data.~~
~~**TX_SEQ_done**: indicates to the SEQ that the texture data transfer is complete for the clause number that is on the TX_SEQ_clause bus.~~

~~**SEQ_SP_phase**: see above under TD1 - shown here for reference.~~
~~**SEQ_SP_write_addr**: see above under TD1- shown here for reference.~~
~~**RF0 write cycle**: see above under TD1- shown here for reference.~~

~~15.~~20. Open issues

Formatted: Bullets and Numbering

There is currently an issue with constants. If the constants are not the same for the whole vector of vertices, we don't have the bandwith from the texture store to feed the ALUs. Two solutions exists for this problem:

1) Let the compiler handle the case and put those instructions in a texture clause so we can use the bandwith there to operate. This requires a significant amount of temporary storage in the register store.
2) Waterfall down the pipe allowing only at a given time the vertices having the same constants to operate in parrallel. This might in the worst case slow us down by a factor of 16.

Need to do some testing on the size of the register file as well as on the register file allocation method (dynamic VS static).

Saving power?

Size of the fifo containing the information of a vector of pixels/vertices. And size of the fifos before the reservation stations.

~~Sequencer Instruction memory, and constant memory.~~

~~Arbitration policy for the output file.~~

Loops and branches.

~~The parameter cache may end up in the PA rather than in the RS. Parameter cache management thus may change.~~

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 2015~~1~~7 ~~October, 20015 October~~	GEN-CXXXXX-REVA	1 of 26

Author: Laurent Lefebvre

Issue To:	**Copy No:**

R400 Sequencer Specification

SEQ

Version 0.8~~7~~

Overview: This is an architectural specification for the R400 Sequencer block (SEQ). It provides an overview of the required capabilities and expected uses of the block. It also describes the block interfaces, internal sub-blocks, and provides internal state diagrams.

AUTOMATICALLY UPDATED FIELDS:
Document Location: C:\perforce\r400\arch\doc\gfx\RE\R400_Sequencer.doc
Current Intranet Search Title: R400 Sequencer Specification

APPROVALS	
Name/Dept	Signature/Date

Remarks:

THIS DOCUMENT CONTAINS CONFIDENTIAL INFORMATION THAT COULD BE SUBSTANTIALLY DETRIMENTAL TO THE INTEREST OF ATI TECHNOLOGIES INC. THROUGH UNAUTHORIZED USE OR DISCLOSURE.

ATI 2014
LG v. ATI
IPR2015-00326

Table Of Contents

1. OVERVIEW..4
1.1 Top Level Block Diagram........................6
1.2 Data Flow graph................................8
1.3 Control Graph..................................9
2. INTERPOLATED DATA BUS...........................9
3. INSTRUCTION STORE..............................12
4. SEQUENCER INSTRUCTIONS.........................12
5. CONSTANT STORE.................................12
6. LOOPING AND BRANCHES...........................12
6.1 The controlling state.........................12
6.2 The Control Flow Program......................13
6.3 Data dependant predicate instructions.........15
6.4 Register file indexing......................16~~15~~
7. REGISTER FILE ALLOCATION.......................16
8. FETCH ARBITRATION............................18~~17~~
9. ALU ARBITRATION..............................18~~17~~
10. HANDLING STALLS.............................18~~17~~
11. CONTENT OF THE RESERVATION STATION FIFOS18~~17~~
12. THE OUTPUT FILE.............................18~~17~~
13. IJ FORMAT...................................19~~17~~
14. THE PARAMETER CACHE.........................19~~18~~
15. VERTEX POSITION EXPORTING...................19~~18~~
16. REAL TIME COMMANDS..........................20~~18~~
17. REGISTERS...................................20~~19~~
18. INTERFACES..................................20~~19~~
18.1 External Interfaces........................20~~19~~
18.1.1 PA/SC to RE : IJ bus.....................20~~19~~
18.1.2 PA/SC to SEQ : IJ Control bus............21~~19~~
18.1.3 VGT to RE : Vertex Bus...................21~~20~~
18.1.4 VGT to SEQ : Vertex Control Bus 21~~20~~
18.1.5 CP to SEQ : Constant store load .21~~20~~
18.1.6 CP to SEQ : Fetch State store load 22~~20~~
18.1.7 CP to SEQ : Control State store load 22~~21~~
18.1.8 MH to SEQ: Instruction store Load 22~~21~~
18.1.9 SP to RB : Pixel read from RBs....22~~21~~
18.1.10 SEQ to RB : Control bus..........22~~21~~
18.1.11 RB to SEQ : Output file control.....22~~21~~
18.1.12 SP to RB : Position return bus......23~~21~~
18.1.13 Shader Engine to Fetch Unit Bus (Fast Bus) 23~~22~~
18.1.14 Sequencer to Fetch Unit bus (Slow Bus) 23~~22~~
19. INTERNAL INTERFACES.........................23~~22~~
20. EXAMPLES OF PROGRAM EXECUTIONS 23~~22~~
20.1.1 Sequencer Control of a Vector of Vertices 23~~22~~
20.1.2 Sequencer Control of a Vector of Pixels 25~~23~~
20.1.3 Notes.....................................25~~24~~
21. OPEN ISSUES.................................26~~24~~
~~1. OVERVIEW..4~~
~~1.1 Top Level Block Diagram........................6~~
~~1.2 Data Flow graph................................8~~
~~1.3 Control Graph..................................9~~
~~2. INTERPOLATED DATA BUS...........................9~~
~~3. INSTRUCTION STORE..............................12~~
~~4. SEQUENCER INSTRUCTIONS.........................12~~
~~5. CONSTANT STORE.................................12~~
~~6. LOOPING AND BRANCHES...........................12~~
~~6.1 The controlling state.........................12~~
~~6.2 The Control Flow Program......................13~~
~~6.3 Data dependant predicate instructions.........14~~
~~6.4 Register file indexing........................14~~
~~7. REGISTER FILE ALLOCATION.......................15~~
~~8. FETCH ARBITRATION..............................16~~
~~9. ALU ARBITRATION................................16~~
~~10. HANDLING STALLS...............................16~~
~~11. CONTENT OF THE RESERVATION STATION FIFOS16~~
~~12. THE OUTPUT FILE...............................16~~
~~13. IJ FORMAT.....................................16~~
~~14. THE PARAMETER CACHE...........................17~~
~~15. VERTEX POSITION EXPORTING.....................17~~
~~16. REAL TIME COMMANDS............................18~~
~~17. REGISTERS.....................................18~~
~~18. INTERFACES....................................18~~
~~18.1 External Interfaces..........................18~~
~~18.1.1 PA/SC to RE : IJ bus.......................18~~
~~18.1.2 PA/SC to SEQ : IJ Control bus..............18~~
~~18.1.3 PA/SC to RE : Vertex Bus...................19~~
~~18.1.4 PA/SC to SEQ : Vertex Control Bus.19~~
~~18.1.5 CP to SEQ : Constant store load......19~~
~~18.1.6 CP to SEQ : Fetch State store load..19~~
~~18.1.7 CP to SEQ : Control State store load 20~~
~~18.1.8 MH to SEQ: Instruction store Load ..20~~
~~18.1.9 SP to RB : Pixel read from RBs........20~~
~~18.1.10 SEQ to RB : Control bus.................20~~
~~18.1.11 SP to PA/SC : Position return bus....20~~
~~18.1.12 Shader Engine to Fetch Unit Bus (Fast Bus) 20~~
~~18.1.13 Sequencer to Fetch Unit bus (Slow Bus) 20~~
~~19. INTERNAL INTERFACES...........................21~~
~~20. EXAMPLES OF PROGRAM EXECUTIONS..21~~
~~20.1.1 Sequencer Control of a Vector of Vertices 21~~
~~20.1.2 Sequencer Control of a Vector of Pixels 22~~
~~20.1.3 Notes.......................................23~~
~~21. OPEN ISSUES...................................23~~
~~1. OVERVIEW..4~~
~~1.1 Top Level Block Diagram........................6~~

~~1.2 Data Flow graph .... 8~~
~~1.3 Control Graph .... 9~~
~~2. INTERPOLATED DATA BUS .... 9~~
~~3. INSTRUCTION STORE .... 12~~
~~4. SEQUENCER INSTRUCTIONS .... 12~~
~~5. CONSTANT STORE .... 12~~
~~6. LOOPING AND BRANCHES .... 12~~
~~6.1 The controlling state .... 12~~
~~6.2 The Control Flow Program .... 13~~
~~6.3 Data dependant predicate instructions .... 14~~
~~6.4 Register file indexing .... 14~~
~~7. REGISTER FILE ALLOCATION .... 15~~
~~8. TEXTURE ARBITRATION .... 16~~
~~9. ALU ARBITRATION .... 16~~
~~10. HANDLING STALLS .... 17~~
~~11. CONTENT OF THE RESERVATION STATION FIFOS17~~
~~12. THE OUTPUT FILE .... 17~~
~~13. THE PARAMETER CACHE .... 17~~
~~14. VERTEX POSITION EXPORTING .... 17~~
~~15. REGISTERS .... 17~~
~~16. INTERFACES .... 18~~
~~16.1 External Interfaces .... 18~~
~~16.1.1 PA/SC to RE : IJ bus .... 18~~
~~16.1.2 PA/SC to SEQ : IJ Control bus .... 18~~
~~16.1.3 PA/SC to RE : Vertex Bus .... 18~~
~~16.1.4 PA/SC to SEQ : Vertex Control Bus . 18~~
~~16.1.5 CP to SEQ : Constant store load ...... 19~~
~~16.1.6 CP to SEQ : Texture State store load 19~~
~~16.1.7 CP to SEQ : Control State store load 19~~
~~16.1.8 MH to SEQ: Instruction store Load .. 19~~
~~16.1.9 SP to RB : Pixel read from RBs .... 19~~
~~16.1.10 SP to PA/SC : Position return bus .... 19~~
~~16.1.11 Shader Engine to Texture Unit Bus (Fast Bus) 19~~
~~16.1.12 Sequencer to Texture Unit bus (Slow Bus) 20~~
~~17. INTERNAL INTERFACES .... 20~~
~~18. EXAMPLES OF PROGRAM EXECUTIONS .. 20~~
~~18.1.1 Sequencer Control of a Vector of Vertices 20~~
~~18.1.2 Sequencer Control of a Vector of Pixels 21~~
~~18.1.3 Notes .... 22~~
~~19. OPEN ISSUES .... 23~~

Revision Changes:

Rev 0.1 (Laurent Lefebvre)
Date: May 7, 2001

Rev 0.2 (Laurent Lefebvre)
Date : July 9, 2001
Rev 0.3 (Laurent Lefebvre)
Date : August 6, 2001
Rev 0.4 (Laurent Lefebvre)
Date : August 24, 2001

Rev 0.5 (Laurent Lefebvre)
Date : September 7, 2001
Rev 0.6 (Laurent Lefebvre)
Date : September 24, 2001
Rev 0.7 (Laurent Lefebvre)
Date : October 5, 2001

Rev 0.8 (Laurent Lefebvre)
Date : October 8, 2001

First draft.

Changed the interfaces to reflect the changes in the SP. Added some details in the arbitration section.
Reviewed the Sequencer spec after the meeting on August 3, 2001.
Added the dynamic allocation method for register file and an example (written in part by Vic) of the flow of pixels/vertices in the sequencer.
Added timing diagrams (Vic)

Changed the spec to reflect the new R400 architecture. Added interfaces.
Added constant store management, instruction store management, control flow management and data dependant predication.
Changed the control flow method to be more flexible. Also updated the external interfaces.

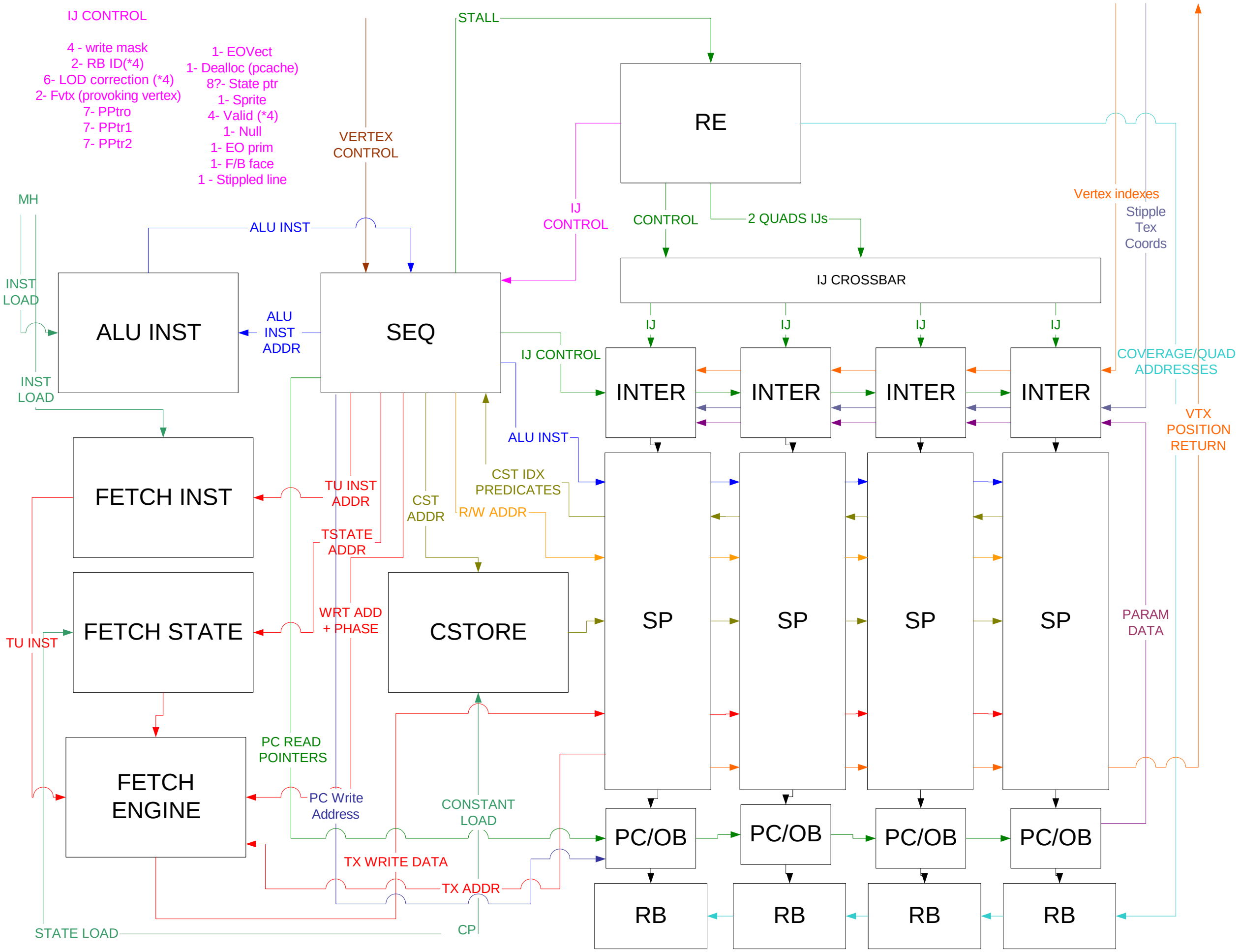
1. Overview

The sequencer first arbitrates between vectors of 64 vertices that arrive directly from primitive assembly and vectors of 16 quads (64 pixels) that are generated in the raster engine.

The vertex or pixel program specifies how many GPR's it needs to execute. The sequencer will not start the next vector until the needed space is available.

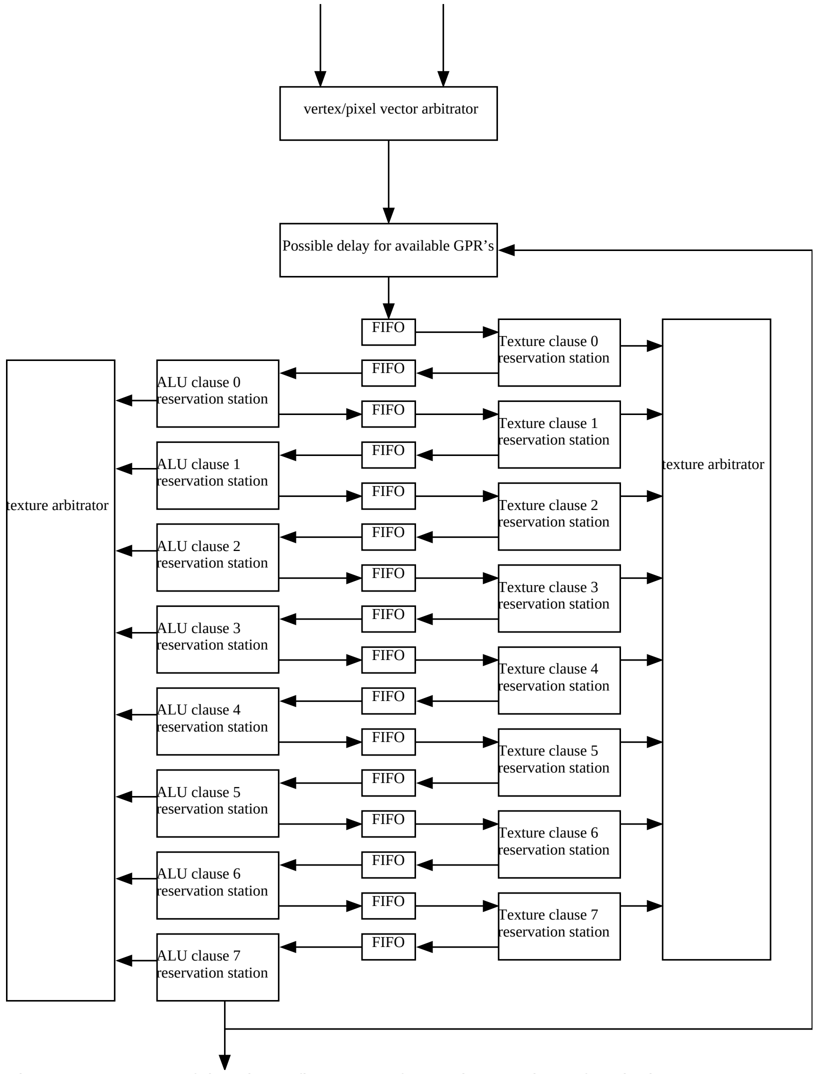
The sequencer is based on the R300 design. It chooses two ALU clauses and a ~~texture~~fetch clause to execute, and executes all of the instructions in a clause before looking for a new clause of the same type. Two ALU clauses are executed interleaved to hide the ALU latency. Each vector will have eight ~~texture~~fetch and eight ALU clauses, but clauses do not need to contain instructions. A vector of pixels or vertices ping-pongs along the sequencer FIFO, bouncing from ~~texture~~fetch reservation station to alu reservation station. A FIFO exists between each reservation stage, holding up vectors until the vector currently occupying a reservation station has left. A vector at a reservation station can be chosen to execute. The sequencer looks at all eight alu reservation stations to choose an alu clause to execute and all eight ~~texture~~fetch stations to choose a ~~texture~~fetch clause to execute. The arbitrator will give priority to clauses/reservation stations closer to the bottom of the pipeline. It will not execute an alu clause until the ~~texture~~fetch fetches initiated by the previous ~~texture~~fetch clause have completed. There are two separate sets of reservation stations, one for pixel vectors and one for vertices vectors. This way a pixel can pass a vertex and a vertex can pass a pixel.

To support the shader pipe the raster engine also contains the shader instruction cache and constant store. There are only one constant store for the whole chip and one instruction store. These will be shared among the four shader pipes. The four shader pipes also execute the same instuction thus there is only one sequencer for the whole chip.



IJ CONTROL
4 - write mask
2- RB ID(*4)
6- LOD correction (*4)
2- Fvtx (provoking vertex)
7- PPtro
7- PPtr1
7- PPtr2
1- EOVect
1- Dealloc (pcache)
8?- State ptr
1- Sprite
4- Valid (*4)
1- Null
1- EO prim
1- F/B face
1 - Stippled line
VERTEX CONTROL
STALL
RE
IJ CONTROL
CONTROL
2 QUADS IJs
Vertex indexes
Stipple Tex Coords
IJ CROSSBAR
MH
ALU INST
INST LOAD
ALU INST
ALU INST ADDR
SEQ
IJ
IJ
IJ
IJ
IJ CONTROL
INTER
INTER
INTER
INTER
COVERAGE/QUAD ADDRESSES
INST LOAD
VTX POSITION RETURN
ALU INST
FETCH INST
TU INST ADDR
CST IDX PREDICATES
CST ADDR
R/W ADDR
TSTATE ADDR
SP
SP
SP
SP
PARAM DATA
TU INST
FETCH STATE
WRT ADD + PHASE
CSTORE
PC READ POINTERS
FETCH ENGINE
PC Write Address
CONSTANT LOAD
PC/OB
PC/OB
PC/OB
PC/OB
TX WRITE DATA
TX ADDR
RB
RB
RB
RB
STATE LOAD
CP

1.1 Top Level Block Diagram



vertex/pixel vector arbitrator
Possible delay for available GPR's
FIFO
Texture clause 0 reservation station
Texture clause 1 reservation station
Texture clause 2 reservation station
Texture clause 3 reservation station
Texture clause 4 reservation station
Texture clause 5 reservation station
Texture clause 6 reservation station
Texture clause 7 reservation station
texture arbitrator
ALU clause 0 reservation station
ALU clause 1 reservation station
ALU clause 2 reservation station
ALU clause 3 reservation station
ALU clause 4 reservation station
ALU clause 5 reservation station
ALU clause 6 reservation station
ALU clause 7 reservation station
texture arbitrator

There are two sets of the above figure, one for vertices and one for pixels.

The rasterizer always checks the vertices FIFO first and if allowed by the sequencer sends the data to the shader. If the vertex FIFO is empty then, the rasterizer takes the first entry of the pixel FIFO (a vector of 64 pixels) and sends it to the interpolators. Then the sequencer takes control of the packet. The packet consists of 21 bits of state, 6-7 bits for the base address of the Shader program and some information on the coverage to determine ~~texture~~fetch LOD. All other information (2x2 adresses) is put in a FIFO (one for the pixels and one for the vertices) and retrieved when the packet finishes its last clause.

On receipt of a packet, the input state machine (not pictured but just before the first FIFO) allocated enough space in the registers to store the interpolated values and temporaries. Following this, the input state machine stacks the packet in the first FIFO.

On receipt of a command, the level 0 ~~texture~~fetch machine issues a texure request and corresponding register address for the ~~texture~~fetch address (ta). A small command (tcmd) is passed to the ~~texture~~fetch system identifying the current level number (0) as well as the register write address for the ~~texture~~fetch return data. One ~~texture~~fetch request is sent every 4 clocks causing the texturing of sixteen 2x2s worth of data (or 64 vertices). Once all the requests are sent the packet is put in FIFO 1.

Upon recept of the return data, the ~~texture~~fetch unit writes the data to the register file using the write address that was provided by the level 0 ~~texture~~fetch machine and sends the clause number (0) to the level 0 ~~texture~~fetch state machine to signify that the write is done and thus the data is ready. Then, the level 0 ~~texture~~fetch machine increments the counter of FIFO 1 to signify to the ALU 1 that the data is ready to be processed.

On receipt of a command, the level 0 ALU machine first decrements the input FIFO counter and then issues a complete set of level 0 shader instructions. For each instruction, the state machine generates 3 source addresses, one destination address (3 cycles later) and an instruction. Once the last instruction as been issued, the packet is put into FIFO 2.

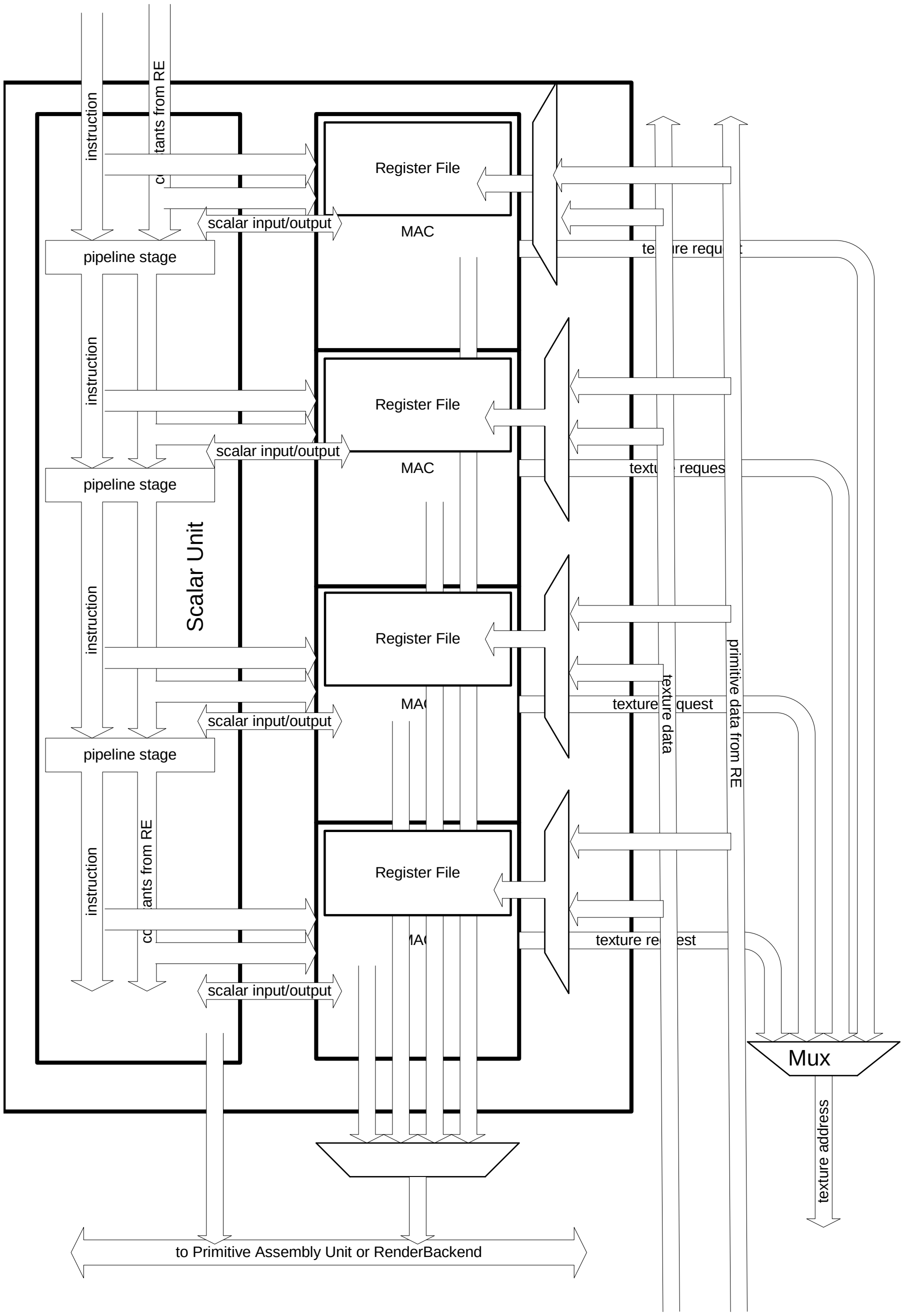
There will always be two active ALU clauses at any given time (and two arbitrers). One arbitrer will arbitrate over the odd clock cycles and the other one will arbitrate over the even clock cycles. The only constraints between the two arbitrers is that they are not allowed to pick the same clause number as they other one is currently working on if the packet os of the same type.

If the packet is a vertex packet, upon reaching ALU clause 4, it can export the position if the position is ready. So the arbitrer must prevent ALU clause 4 to be selected if the positional buffer is full (or can't be accessed). Along with the positional data, the location where the vertex data is to be put is also sent (parameter data pointers).

All other level process in the same way until the packet finally reaches the last ALU machine (8). On completion of the level 8 ALU clause, a valid bit is sent to the Render Backend which picks up the color data. This requires that the last instruction writes to the output register – a condition that is almost always true. If the packet was a vertex packet, instead of sending the valid bit to the RB, it is sent to the PA so it can know that the data present in the parameter store is valid.

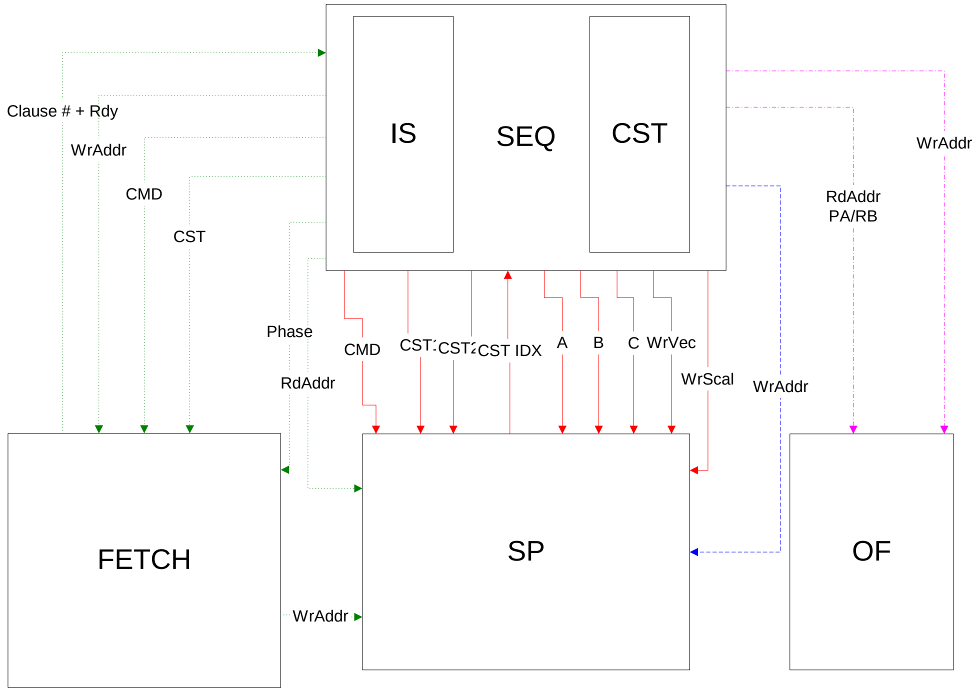
Only two ALU state machine may have access to the register file address bus or the instruction decode bus at one time. Similarly, only one ~~texture~~fetch state machine may have access to the register file address bus at one time. Arbitration is performed by three arbitrer blocks (two for the ALU state machines and one for the ~~texture~~fetch state machines). The arbitrers always favor the higher number state machines, preventing a bunch of half finished jobs from clogging up the register files.

1.2 Data Flow graph



The gray area represents blocks that are replicated 4 times per shader pipe (16 times on the overall chip).

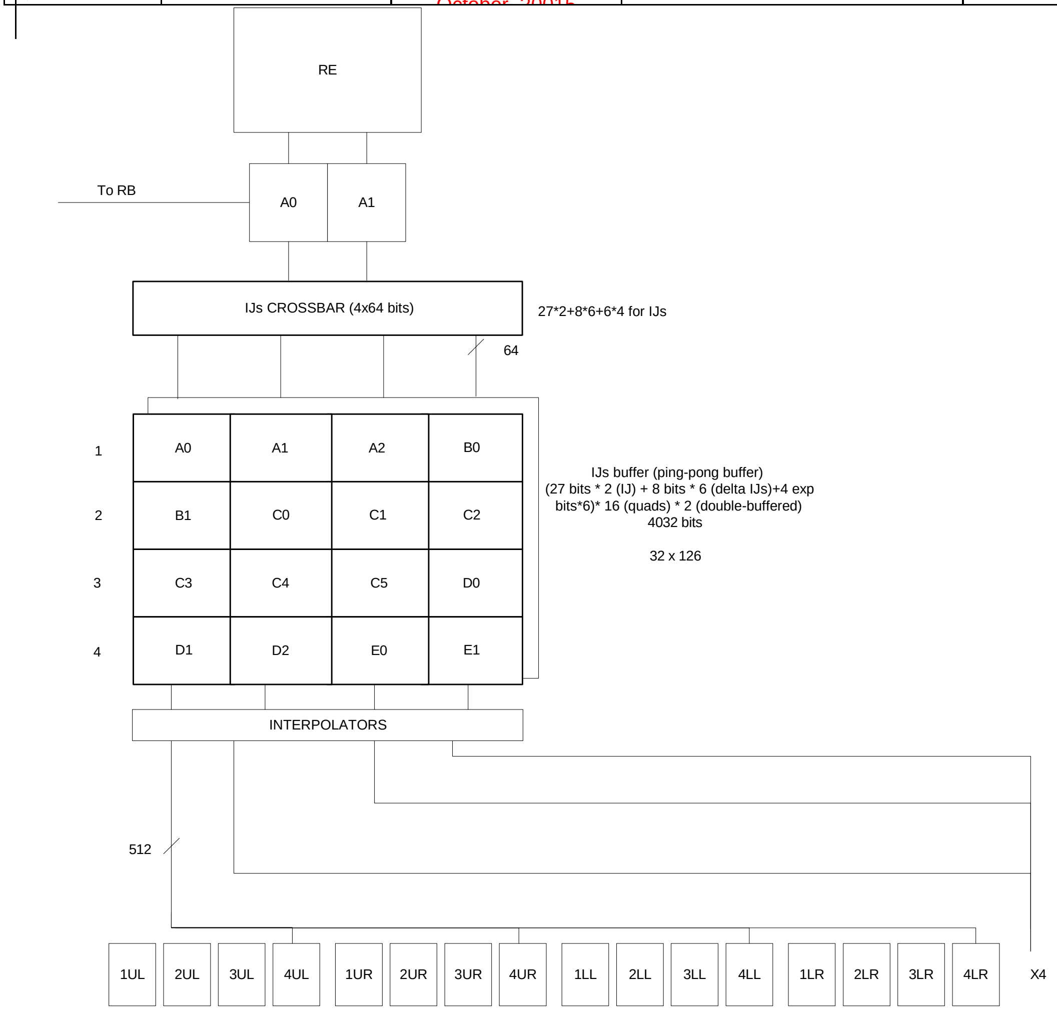
1.3 Control Graph



In green is represented the ~~Texture~~Fetch control interface, in red the ALU control interface, in blue the Interpolated/Vector control interface and in purple is the output file control interface.

2. Interpolated data bus

The interpolators contain an IJ buffer to pack the information as much as possible before writing it to the register file.


RE
To RB
A0
A1
IJs CROSSBAR (4x64 bits)
27*2+8*6+6*4 for IJs
64
1
2
3
4
A0
A1
A2
B0
B1
C0
C1
C2
C3
C4
C5
D0
D1
D2
E0
E1
IJs buffer (ping-pong buffer)
(27 bits * 2 (IJ) + 8 bits * 6 (delta IJs)+4 exp bits*6)* 16 (quads) * 2 (double-buffered)
4032 bits
32 x 126
INTERPOLATORS
512
1UL
2UL
3UL
4UL
1UR
2UR
3UR
4UR
1LL
2LL
3LL
4LL
1LR
2LR
3LR
4LR
X4

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19
SP0	A0	B1	C3	D1					A0	B1	C3	D1					V 0-3	V 16- 19	V 32- 35	V 48- 51
SP1	A1		C4	D2		C0			A1		C4	D2		C0			V 4-7	V 20- 23	V 36- 39	V 52- 55
SP2	A2		C5			C1		E0	A2		C5			C1		E0	V 8-11	V 24- 27	V 40- 43	V 56- 59
SP3					B0	C2	D0	E1					B0	C2	D0	E1	V 12- 15	V 28- 31	V 44- 47	V 60- 63
	P0								P1											

Above is an example of a tile we might receive. The IJ information is packed in the IJ buffer 2 quads at a time. The sequencer allows at any given time as many as four quads to interpolate a parameter. They all have to come from the same primitive. Then the sequencer controls the write mask to the register to write the valid data in.

3. Instruction Store

There is going to be only one instruction store for the whole chip. It will contain 4096 instructions of 96 bits each. There is also going to be a control instruction store of size 256(512?)x32.

{ISSUE : The instruction store is loaded by the sequencer using the memory hub ?}.

The read bandwith from this store is 96*2 bits/ 4 clocks (48 bits/clock). It is likely to be a 1~~R/1W~~ port memory; we use ~~2~~ 1 clock~~s~~ to load the ALU instruction, ~~and~~ 1~~2~~ clocks to load the ~~Texture~~Fetch instruction, 1 clock to load 2 control flow instructions and 1 clock to write instructions.

4. Sequencer Instructions

All control flow instructions and move instructions are handled by the sequencer only. The ALUs will perform NOPs during this time (MOV PV,PV, PS,PS).

5. Constant Store

The constant store is managed by the CP. The sequencer is aware of where the constants are using a remaping table also managed by the CP. A likely size for the constant store is 512x128 bits. The constant store is also planned to be shared. The read BW from the constant store is 128 bits/clock and the write bandwith is 32/4 bits/clock.

In order to do constant store indexing, the sequencer must be loaded first with the indexes (that come from the GPRs). There are 144 wires from the exit of the SP to the sequencer (9 bits pointers x 16 vertexes/clock). Since the data must pass thru the Shader pipe for the float to fixed convertion, there is a latency of 4 clocks (1 instruction) between the time the sequencer is loaded and the time one can index into the constant store. The assembly will look like this

```
MOVA  R1.X,R2.X        // Loads the sequencer with the content of R2.X, also copies the content of R2.X into R1.X
NOP                    // latency of the float to fixed conversion
ADD   R3,R4,C0[R2.X]// Uses the state from the sequencer to add R4 to C0[R2.X] into R3
```

Note that we don't really care about what is in the brackets because we use the state from the MOVA instruction. R2.X is just written again for the sake of simplicity.

The storage needed in the sequencer in order to support this feature is 2*64*9 bits = 1152 bits.

6. Looping and Branches

Loops and branches are planned to be supported and will have to be dealt with at the sequencer level. We plan on supporting constant loops and branches using a control program. The control program has 4(5) instructions:

6.1 The controlling state.

As per Dx9 the following state is available for control flow:

Boolean[15:0]
loop_count[7:0][7:0]
In addition:
loop_start [7:0] [7:0]
loop_step [7:0] [7:0]
Exist to give more control to the controlling program.

We will extend that in the R400 to:

Boolean[255~~31~~:0]
Loop_count[7:0][15:0]
Loop_Start[7:0] [15:0]
Loop_End[7:0] [15:0]

{ISSUE: How is the controlling state loaded and how many contexts do we have?}

6.2 The Control Flow Program

The R300 uses a match method for control flow: The shader is executed, and at every instruction its address is compared with addresses (or address?) in a control table. The "event" in the control table can redirect operations in the program.

The Method chosen for the R400 is a "control program". The control program has ~~four~~ ten basic instructions:

Execute
Conditional_execute
~~(~~Conditional_~~-~~Execut~~e~~e_~~-~~Predicates~~)~~
Conditional_execute_or_Jump
Conditional_jump
Call
Return
Loop_start
Loop_end
End_of_clause

Execute, causes the specified number of instructions in instruction store to be executed.
Conditional_execute checks a condition first, and if true, causes the specified number of instructions in instruction store to be executed.
Loop_start resets the corresponding loop counter to the start value on the first pass after it checks for the end condition and if met jumps over to a specified address.
Loop_end increments (decrements?) the loop counter and jumps back the specified number of instructions.
Call jumps to an address and pushes the IP counter on the stack. On the return instruction, the IP is poped from the stack.
Conditional_execute_or_Jump executes a block of instructions or jumps to an address is the condition is not met.
Conditional_execute_Predicates executes a block of instructions if all bits in the predicate vectors meet the condition.
End_of_clause marks the end of a clause.
Conditional_jumps jumps to an address if the condition is met. ~~if the loop end condition is not met.~~

~~if we try and fit the control flow instructions into 32 bit words, the following instructions are possible choices:~~We have to fit instructions into 48 bits in order to be able to put two control flow instruction per line in the instruction store.

Execute				
47	46… 42	41 … 24	23 … 12	11 … 0
Addressing	00001	RESERVED	Instruction_count	Exec Address

Execute up to ~~4K~~ 4k instructions at the specified address in the instruction memory.

Conditionnal_Execute_or_Jump						
47	46 … 42	41 … 34	33	32 … 21	20 … 12	11 … 0
Addressing	00010	Booleans	Condition	Jump address	Instruction_count	Exec Address

I~~i~~f the specified boolean (~~6~~8 bits can address ~~64~~256 booleans) meets the specified condition then execute the specified instructions (up to ~~64~~ 512 instructions) or if the condition is not met jump to the jump address in the control flow program. This MUST be a forward jump.

Conditionnal_Execute						
47	46 … 42	41 … 34	33	32 … 21	20 … 12	11 … 0
Addressing	00011	Boolean address	Condition	RESERVED	Instruction_count	Exec Address

If the specified boolean (8 bits can address 256 booleans) meets the specified condition then execute the specified instructions (up to 512 instructions)

Conditionnal_Execute_Predicates						
47	46 … 42	41 … 38	37	36 … 21	20 … 12	11 … 0
Addressing	00100	Predicate vector	Condition	RESERVED	Instruction_count	Exec Address

Check the OR of all current predicate bits. If OR matches the condition execute the specified number of instructions.

~~Initialize the specified loop~~

~~If the loop condition of the current loop is not met, then branch back to the specified address in the control flow program. Note that jumping back to the loop_start results in an infinite loop, the jump should be to loop_start+1.~~

Loop_Start				
47	46 … 42	41 … 16	15 … 4	3 … 0
Addressing	00101	RESERVED	Jump address	Loop ID

Loop Start. Compares the loop count with the end value. If loop condition not met jump to the address. Forward jump only. Also computes the index value.

Loop_End				
47	46 … 42	41 … 16	15 … 4	3 … 0
Addressing	00111	RESERVED	Start address	Loop ID

Loop end. Increments the counter by one and jumps BACK only to the start of the loop.

The way this is described does not prevent nested loops, and the inclusion of the loop id make this easy to do.

Call			
47	46 … 42	41…12	11 … 0
Addressing	01000	RESERVED	Address

Jumps to the specified address and pushes the IP counter on the stack.

Return		
47	46 … 42	41 … 0

Addressing	01001	RESERVED

Pops the topmost address from the stack and jumps to that address.

Conditionnal_Jump					
47	46 … 42	41 … 34	33	32 … 12	11 … 0
Addressing	01010	Boolean address	Condition	RESERVED	Address

If condition met, jumps to the address. FORWARD jump only allowed.

End_of_Clause		
47	46 … 42	41 … 0
Addressing	01011	RESERVED

Marks the end of a clause.

To prevent infinite loops, we will keep 9 bits loop counters instead of 8 (we are only able to loop 256 times). If the counter goes higher than 255 then the loop_end or the loop_start is going to break the loop. The sequencer will keep a loop index value of 17 bits. This will be updated everytime we loop and can only be used to index the constant store and the register file. The way to compute this value is:

Index = Loop_counter*Loop_iterator + Loop_init.

The basic model is as follows:

The render state defined the clause boundaries:

```
Vertex_shader_fetch[7:0][7:0]    // eight 8 bit pointers to the location where each clauses control program is located
Vertex_shader_alu[7:0][7:0]      // eight 8 bit pointers to the location where each clauses control program is located
Pixel_shader_fetch[7:0][7:0]     // eight 8 bit pointers to the location where each clauses control program is located
Pixel_shader_alu[7:0][7:0]       // eight 8 bit pointers to the location where each clauses control program is located
```

~~The control program can be up to 256 instructions in size. (There is an offset added to the address from the render state before accessing the control flow program memory to allow for multiple programs resident at the same time)~~

The control program for a given clause is executed to completion before moving to another clause, (with the exception of the pick two nature of the alu execution). The control program is the only program aware of the clause boundaries.

~~The addresses from the control program are added to another offset to allow for multiple programs resident at the same time.~~

~~Under this model, all subroutine calls must be inlined into the control program.~~

6.3 Data dependant predicate instructions

Data dependant conditionals will be supported in the R400. The only way we plan to support those is by supporting three vector/scalar predicate operations of the form:

PRED_SETE_# - similar to SETE except that the result is 'exported' to the sequencer.

PRED_SETGT_# - similar to SETGT except that the result is 'exported' to the sequencer
PRED_SETGTE_# - similar to SETGTE except that the result is 'exported' to the sequencer

For the scalar operations only we will also support the two following instructions:
PRED_SETE0_# – SETE0
PRED_SETE1_# – SETE1

The export is a single bit - 1 or 0 that is sent using the same data path as the MOVA instruction. The sequencer will maintain ~~the~~4 sets of 64 bit predicate vectors (in fact 8 sets because we interleave two programs but only 4 will be exposed) and use it to control the write masking~~ (two sets for interleaved operation)~~. This predicate is not maintained across clause boundaries. The # sign is used to specify wich predicate set you want to use 0 thru 3.

Then we have two conditional execute bits. The first bit is a conditional execute “on” bit and the second bit tells us if we execute on 1 or 0. For exemple, the instruction :

P0_ADD_# R0,R1,R2

Is only going to write the result of the ADD into those GPRs whose predicate bit is 0. Alternatively, P1_ADD_# would only write the results to the GPRs whose predicate bit is set. The use of the P0 or P1 without precharging the sequencer with a PRED instruction is undefined.

{Issue: do we have to have a NOP between PRED and the first instruction that uses a predicate?}

6.4 Register file indexing

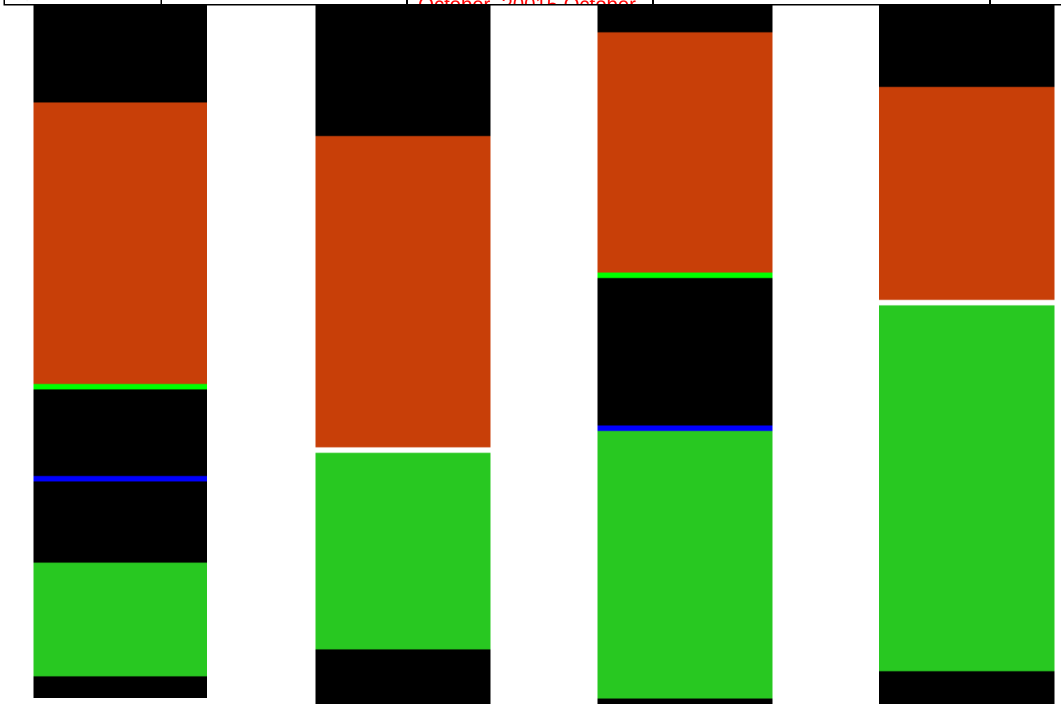
Because we can have loops in ~~texture~~fetch clause, we need to be able to index into the register file in order to retrieve the data created in a ~~texture~~fetch clause loop and use it into an ALU clause. The instruction will include the base address for register indexing and the instruction will contain these controls :

Bit7	Bit 6	
0	0	'absolute register'
0	1	'relative register'
1	0	'previous vector'
1	1	'previous scalar'

In the case of an absolute register we just take the address as is. In the case of a relative register read we take the base address and we add to it the ~~loop_counter~~loop_index and this becomes our new address that we give to the shader pipe.

7. Register file allocation

The register file allocation for vertices and pixels can either be static or dynamic. In both cases, the register file in managed using two round robins (one for pixels and one for vertices). In the dynamic case the boundary between pixels and vertices is allowed to move, in the static case it is fixed to VERTEX_REG_SIZE for vertices and 256-VERTEX_REG_SIZE for pixels.



Above is an example of how the algorithm works. Vertices come in from top to bottom; pixels come in from bottom to top. Vertices are in orange and pixels in green. The blue line is the tail of the vertices and the green line is the tail of the pixels. Thus anything between the two lines is shared. When pixels meets vertices the line turns white and the boundary is static until both vertices and pixels share the same “unallocated bubble”. Then the boundary is allowed to move again.

8. ~~Texture~~Fetch Arbitration

The ~~texture~~fetch arbitration logic chooses one of the 8 potentially pending ~~texture~~fetch clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute. Once chosen, the clause state machine will send one 2x2 ~~texture~~ fetch per clock (or 4 fetches in one clock every 4 clocks) until all the ~~texture~~ fetch instructions of the clause are sent. This means that there cannot be any dependencies between two ~~texture~~ fetches of the same clause.

The arbitrator will not wait for the ~~texture~~ fetches to return prior to selecting another clause for execution. The ~~texture~~fetch pipe will be able to handle up to X(?) in flight ~~texture~~ fetches and thus there can be a fair number of active clauses waiting for their ~~texture~~fetch return data.

9. ALU Arbitration

ALU arbitration proceeds in almost the same way than ~~texture~~fetch arbitration. The ALU arbitration logic chooses one of the 8 potentially pending ALU clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute. There are two ALU arbitrers, one for the even clocks and one for the odd clocks. For exemple, here is the sequencing of two interleaved ALU clauses (E and O stands for Even and Odd sets of 4 clocks):

Einst0 Oinst0 Einst1 Oinst1 Einst2 Oinst2 Einst0 Oinst3 Einst1 Oinst4 Einst2 Oinst0…
Proceeding this way hides the latency of 8 clocks of the ALUs.

10. Handling Stalls

When the output file is full, the sequencer prevents the ALU arbitration logic to select the last clause (this way nothing can exit the shader pipe until there is place in the output file. If the packet is a vertex packet and the position buffer is full (POS_FULL) then the sequencer also prevents a thread to enter the exporting clause (4?). The sequencer will set the OUT_FILE_FULL signal n clocks before the output file is actually full and thus the ALU arbitrer will be able read this signal and act accordingly by not preventing exporting clauses to proceed.

11. Content of the reservation station FIFOs

3 bits of Render State 6-7 bits for the base address of the instruction store, some bits for LOD correction and coverage mask information in order to fetch ~~texture~~fetch for only valid pixels. Every other information (such as the coverage mask, quad address, etc.) is put in a FIFO and is retrieved when the quad exits the shader pipe to enter in the output file buffer. Since pixels and vertices are kept in order in the shader pipe, we only need two fifos (one for vertices and one for pixels) deep enough to cover the shader pipe latency. This size will be determined later when we will know the size of the small fifos between the reservation stations.

12. The Output File

The output file is where pixels are put before they go to the RBs. The write BW to this store is 256 bits/clock. Just before this output file are staging registers with write BW 512 bits/clock and read BW 256 bits/clock. For this reason only ONE concurrent program can be of clause 8 (exporting clause) the other program MUST not. The staging registers are 4x128 (and there are 16 of those on the whole chip).

13. IJ Format

The IJ information sent by the PA is of this format on a per quad basis:

We have a vector of IJ's (one IJ per pixel at the centroid of the fragment or at the center of the pixel depending on the mode bit). The interpolation is done at a different precision across the 2x2. The upper left pixel's parameters are always interpolated at full 19~~24~~x24 mantissa precision. Then the result of the interpolation along with the difference in IJ in reduced precision is used to interpolate the parameter for the other three pixels of the 2x2. Here is how we do it:

Assuming P0 is the interpolated parameter at Pixel 0 having the barycentric coordinates I(0), J(0) and so on for P1,P2 and P3. Also assuming that A is the parameter value at V0 (interpolated with I), B is the parameter value at V1 (interpolated with J) and C is the parameter value at V2 (interpolated with (1-I-J).

$$\Delta 01I = I(1) - I(0)$$
$$\Delta 01J = J(1) - J(0)$$
$$\Delta 02I = I(2) - I(0)$$
$$\Delta 02J = J(2) - J(0)$$
$$\Delta 03I = I(3) - I(0)$$
$$\Delta 03J = J(3) - J(0)$$

P0	P1
P2	P3

$$P0 = C + I(0)*(A-C) + J(0)*(B-C)$$
$$P1 = P0 + \Delta 01I*(A-C) + \Delta 01J*(B-C)$$
$$P2 = P0 + \Delta 02I*(A-C) + \Delta 02J*(B-C)$$
$$P3 = P0 + \Delta 03I*(A-C) + \Delta 03J*(B-C)$$

P0 is computed at ~~full~~ 19~~24~~x24 mantissa precision and P1 to P3 are computed at 8X24 mantissa precision. So far no visual degradation of the image was seen using this scheme.

Multiplies (Full Precision): 2
Multiplies (Reduced precision): 6
~~Subtracts 24x24 yielding 8 bits (IJs): 6~~
Subtracts ~~24x24~~ 19x24 (Parameters): 2
Adds: 8

FORMAT OF P0's IJ : Mantissa ~~23~~19 Exp 4 for I + Sign
Mantissa ~~23~~19 Exp 4 for J + Sign

FORMAT of Deltas (x3): Mantissa 8 Exp 4 for I + Sign
Mantissa 8 Exp 4 for J + Sign

Total number of bits : 19~~23~~*2 + 8*6 + 4*8 + 4*2 = 126 ~~(rounded up on the bus to 128)~~

14. The parameter cache

The parameter cache is where the vertex shaders export their data. It consists of 16 128x128 memories (1R/1W). The reuse engine will make it so that all vertexes of a given primitive will hit different memories.

15. Vertex position exporting

On clause 4 (or 5) the vertex shader can export to the PA both the vertex position and the point sprite. It can also do so at clause 8 if not done at clause 4. ~~The export is done by putting the exported position back into the GPRs. Then using the texture port in an opportunistic manner, 16 positions are put into a FIFO (16x128) in order (left to right). This fifo drains 128 bits per clock to the PA and once empty is filled up again with sprite sizes (if any). The process is~~

~~repeated 4 times. The sequencer must make sure that the program doesn't enter ALU clause 5 (it can enter texture clause 5) because the registers can be reused at this point. The sequencer must also make sure not to dealocate an exporting program before it is done exporting data.~~ Along with the position is exported a pointer to the parameter cache where the data will be once the vertex shader exports. The storage needed to perform the position export is at least 64x128 memories for the position and 64x32 memories for the sprite size. It is going to be taken in the pixel output fifo.

Formatted: Bullets and Numbering

16. Real time commands

We are unable to use the parameter memory since there is no way for a command stream to write into it. Instead we need to add three 16x128 memories (one for each of three vertices x 16 interpolants). These will be mapped onto the register bus and written by type 0 packets, and output to the the parameter busses (the sequencer and/or PA need to be able to address the realtime parameter memory as well as the regular parameter store. For higher performance we should be able able to view them as two banks of 16 and do double buffering allowing one to be loaded, while the other is rasterized with. Most overlay shaders will need 2 or 4 scalar coordinates, one option might be to restrict the memory to 16x64 or 32x64 allowing only two interpolated scalars per cycle, the only problem I see with this is, if we view support for 16 vector-4 interpolants important (true only if we map microsoft's high priority stream to the realtime stream), then the PA/sequencer need to support a realtime-specific mode where we need to address 32 vectors of parameters instead of 16.

Formatted: Bullets and Numbering

~~16.~~17. Registers

DYNAMIC_REG	Dynamic allocation (pixel/vertex) of the register file on or off.
VERTEX_REG_SIZE	What portion of the register file is reserved for vertices (static allocation only)
PIXEL_MIN_SIZE	Minimal size of the register file's pixel portion (dynamic only)
VERTEX_MIN_SIZE	Minimal size of the register file's vertex portion (dynamic only)
Vshader_fetch[~~7~~11:0][7:0]	eight ~~8~~ 12 bit pointers to the location where each clauses control program is located
Vshader_alu[~~7~~11:0][7:0]	eight ~~8~~ 12 bit pointers to the location where each clauses control program is located
Pshader_fetch[~~7~~11:0][7:0]	eight ~~8~~ 12 bit pointers to the location where each clauses control program is located
Pshader_alu[~~7~~11:0][7:0]	eight ~~8~~ 12 bit pointers to the location where each clauses control program is located
PSHADER	base pointer for the pixel shader
VSHADER	base pointer for the vertex shader
~~PCNTLSHADER~~	~~base pointer for the pixel control program~~
~~VCNTLSHADER~~	~~base pointer for the vertex control program~~
VWRAP	wrap point for the vertex shader instruction store
PWRAP	wrap point for the pixel shader instruction store
REG_ALLOC_PIX	number of registers to allocate for pixel shader programs
REG_ALLOC_VERT	number of registers to allocate for vertex shader programs
PARAM_MASK[0…16]	parameter mask to specify ~~wich~~ how parameters maps in the pixel shader
FLAT_GOUR[0…16]	wich parameters are to be gouraud shaded
GEN_TEX[0….16]	for wich parameters do we need to generate tex coords.
CYL_WRAP[0…64]	for wich ~~vertices~~ parameters (and channels (xyzw)) do we do the cyl wrapping.
P_EXPORT	number of exports for pixel shader
V_EXPORT	number of exports for vertex shader (also the number of interpolated parameters for pixel shaders)
V_EXPORT_LOC	Vertex shader exporting to RB or the PCACHE
ARBITRATION_policy	policy of the arbitration between vetexes and pixels

Formatted: Bullets and Numbering

~~17.~~18. Interfaces

~~17.1~~18.1 External Interfaces

~~17.1.1~~18.1.1 *PA/SC to RE : IJ bus*

This is a bus that sends the IJ information to the IJ fifos on the top of each shader pipe. At the same time the control information goes to the sequencer

Name	Direction	Bits	Description

IJs	PA→RE	6~~3~~4	IJ information sent over 2 clocks
Mask	PA→RE	1	Write Mask

~~17.1.2~~18.1.2 *PA/SC to SEQ : IJ Control bus*

Formatted: Bullets and Numbering

This is the control information sent to the sequencer in order to control the IJ fifos and all other information needed to execute a shader program on the sent pixels.

Name	Direction	Bits	Description
Write Mask	PA→SEQ(RE)	4	Quad Write mask left to right
RB_ID	PA→SEQ(RE)	8	RB id for each quad sent 2 bits per quad
LOD_CORRECT	PA→SEQ(RE)	24	LOD correction per quad (6 bits per quad)
FVTX	PA→SEQ(RE)	2	Provoking vertex for flat shading
PPTR0	PA→SEQ(RE)	11	P Store pointer for vertex 0
PPRT1	PA→SEQ(RE)	11	P Store pointer for vertex 1
PPTR2	PA→SEQ(RE)	11	P Store pointer for vertex 2
E_OFF_VECTOR	PA→SEQ(RE)	1	End of the vector
DEALLOC	PA→SEQ(RE)	1	Deallocation token for the P Store
STATE	PA→SEQ(RE)	21	State/constant pointer (6*3+3)
VALID	PA→SEQ(RE)	16	Valid bits for all pixels
NULL	PA→SEQ(RE)	1	Null Primitive (for PC deallocation purposes)
E_OFF_PRIM	PA→SEQ(RE)	1	End Of the primitive
FBFACE	PA→SEQ(RE)	1	Front face = 1, back face = 0
~~STIPPLE_LINE~~TYPE	PA→SEQ(RE)	3~~1~~	Stippled line and Real time command need to load tex cords from alternate buffer 000 : Normal 001 : Stippled line 011 : Real Time 100 : Line AA 101 : Point AA 110 : Sprite
RTRn	SEQ→PA	1	Stalls the PA in n clocks
RTS	PA→SEQ(RE)	1	PA ready to send data
QuadX	PA→SEQ(RE)	~~40~~8	Quad X address ~~10~~ 2 bits per quad
QuadY	PA→SEQ(RE)	~~40~~8	Quad Y address ~~10~~ 2 bits per quad

~~17.1.3~~18.1.3 *~~PA/S~~VGT~~C~~ to RE : Vertex Bus*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
Vertex indexes	VGT~~PA~~→RE	~~32~~128	Pointers of indexes or HOS surface information
EOF_vector	VGT→RE	1	End of the vector
Inputs_vert	VGT→RE	1	0: Normal 128 bits per vert 1: double 256 bits per vert

~~17.1.4~~18.1.4 *VGT~~PA/SC~~ to SEQ : Vertex Control Bus*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
STATE	VGT~~PA~~→SEQ	21	Render State (6*3+3 for constants)
~~Write Mask~~Vert counter	VGT~~PA~~→SEQ	~~64~~?6	Which vertices are valid
Inputs_vert	VGT→SEQ	1	0: Normal 128 bits per vert 1: double 256 bits per vert

This information needs to be sent over 64 clocks.

~~17.1.5~~18.1.5 *CP to SEQ : Constant store load*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
Constant Address	CP→SEQ	8	Address of the block of 4 constants

Constant Data	CP→SEQ	512	Data sent over 4 clocks
Remap Address	CP→SEQ	10	Remaping address write address
Remap Data pointer	CP→SEQ	8	Remaping pointer

~~17.1.6~~18.1.6 *CP to SEQ : ~~Texture~~Fetch State store load*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
Constant Address	CP→SEQ	8	Address of the block of 4 state constants
Constant Data	CP→SEQ	512	Data sent over 4 clocks
Remap Address	CP→SEQ	10	Remaping address write address
Remap Data pointer	CP→SEQ	8	Remaping pointer

~~17.1.7~~18.1.7 *CP to SEQ : Control State store load*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description

{ISSUE: How,Who and what is the size of this bus?}

~~17.1.8~~18.1.8 *MH to SEQ: Instruction store Load*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
Instruction address	MH→SEQ	12	Instruction address
Instruction	MH→SEQ	96	Instruction X times
Control Instruction address	MH→SEQ	9	Pointer to the control instruction store
Control Instruction	MH→SEQ	32	Control Instruction X times

~~17.1.9~~18.1.9 *SP to RB : Pixel read from RBs*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
~~Pixel Data~~Export_data	SP→RB	~~256~~64	~~2 pixels (or ½ quad)~~a pair of 32 bits channel values
ExportID	SP→RB	9	0cvvvvhqq: Vertex data vvvv 0-15 from first or second clause (c=0 or 1), XY or ZW components (h=0 or 1), quad 0-3 in the shader (qq= 0-3) 1cbbkttqq: Pixel data for buffer bb (0-3) from first or second clause (0-1) killed or not (k=1 or 0) quad 0-3 in the shader and data is RG (tt=0), BA (tt=1) or Z (tt=2)
ExportMask	SP→RB	2	Specifies whether to write low, high or both 32 bit words. If export mask is 00 data is invalid
ExportLast	SP→RB	1	Last export instruction of the clause

18.1.10 *SEQ to RB : Control bus*

Formatted: Bullets and Numbering

Name	**Direction**	**Bits**	**Description**
Type	SEQ→RB	1	0: Pixel 1: Vertex
Interleaving	SEQ→RB	1	0: first interleaved clause 1: second interleaved clause
Export_size	SEQ→RB	4	0 thru 16 parameters exported for vertexes (vvvv) OR (bbzs) 1-4 color buffers (bb), two component (s=0) or 4 component colors (s=1) with z (z=1) or without z (z=0)
Valid	SEQ→RB	1	Data valid

Only one exporting clause (7) can be selected at any given time.

18.1.11 *RB to SEQ : Output file control*

Formatted: Bullets and Numbering

Name	**Direction**	**Bits**	**Description**
Buff_Full	RB→SEQ	1	Set if full

Avail_size	RB→SEQ	6	Size available in output buffers (in 32bits increments)

~~17.1.10~~18.1.12 *SP to ~~PA/SC~~RB : Position return bus*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
Position return	SP→~~PA~~RB	128	Position data or sprite size (per clock)
Parameter cache pointer	SP→~~PA~~RB	11	Pointer where the data will be in the parameter cache for each vertex

For point sprites and position exports the size and position are interleaved on a 16 x 16 basis. We export ~~16~~1 position~~s~~ then ~~16~~ 1 point sprite sizes. The storage used is of 64x128 bits for position and 64x32 bits for sprite size, it is taken from the output buffer. Additionnally,if needed the edge flags are packed into the bits of the sprite sizes.~~The registers are taken until the next ALU clause where they are going to be available again. Thus the sequencer has to make sure that we finished exporting data before allowing the program in the next ALU clause.~~

~~17.1.11~~18.1.13 *Shader Engine to ~~Texture~~Fetch Unit Bus (Fast Bus)*

Formatted: Bullets and Numbering

Four quad's worth of addresses is transferred to ~~Texture~~Fetch Unit every clock. These are sourced from a different pixel within each of the sub-engines repeating every 4 clocks. The register file index to read must precede the data by 2 clocks. The Read address associated with Quad 0 must be sent 1 clock after the Instruction Start signal is sent, so that data is read 3 clocks after the Instruction Start.

Four Quad's worth of ~~Texture~~Fetch Data may be written to the Register file every clock. These are directed to a different pixel of the sub-engines repeating every 4 clocks. The register file index to write must accompany the data. Data and Index associated with the Quad 0 must be sent 3 clocks after the Instruction Start signal is sent.

Name	Direction	Bits	Description
Tex_Read_Register_Index	SEQ->SP	7	Index into Register files for reading ~~Texture~~Fetch Address
Tex_RegFile_Read_Data	SP->TEX	2048	16 ~~Texture~~Fetch Addresses read from the Register file
Tex_Write_Register_Index	SEQ->TEX	7	Index into Register file for write of returned ~~Texture~~Fetch Data

~~17.1.12~~18.1.14 *Sequencer to ~~Texture~~Fetch Unit bus (Slow Bus)*

Formatted: Bullets and Numbering

Once every four clock, the ~~texture~~fetch unit sends to the sequencer on wich clause it is now working and if the data in the registers is ready or not. This way the sequencer can update the ~~texture~~fetch counters for the reservation station fifos. The sequencer also provides the intruction and constants for the~~ texture~~ fetch to execute and the address in the register file where to write the ~~texture~~fetch return data.

Name	Direction	Bits	Description
Tex_Ready	TEX→ SEQ	1	Data ready
Tex_Clause_Num	TEX→ SEQ	3	Clause number
Tex_cst	SEQ→TEX	10	~~Texture~~Fetch state address 10 bits sent over 4 clocks
Tex_Inst	SEQ→TEX	12	~~Texture f~~Fetch instruction address 12 bits sent over 4 clocks
EO_CLAUSE	SEQ→TEX	1	Last instruction of the clause
PHASE	SEQ→TEX	1	Write phase signal

Formatted: Bullets and Numbering

~~18.~~19. Internal interfaces

~~19.~~20. Examples of program executions

~~19.1.1~~20.1.1 *Sequencer Control of a Vector of Vertices*

1. PA sends a vector of 64 vertices (actually vertex indices – 32 bits/index for 2048 bit total) to the RE's Vertex FIFO
   - state pointer as well as tag into position cache is sent along with vertices
   - space was allocated in the position cache for transformed position before the vector was sent

- **also before the vector is sent to the RE, the CP has loaded the global instruction store with the vertex shader program (using the MH?)**
- The vertex program is assumed to be loaded when we receive the vertex vector.
    - the SEQ then accesses the IS base for this shader using the local state pointer (provided to all sequencers by the RBBM when the CP is done loading the program)

2. SEQ arbitrates between the Pixel FIFO and the Vertex FIFO – basically the Vertex FIFO always has priority
    - at this point the vector is removed from the Vertex FIFO
    - the arbitrer is not going to select a vector to be transformed if the parameter cache is full unless the pipe as nothing else to do (ie no pixels are in the pixel fifo).
3. SEQ allocates space in the SP register file for index data plus GPRs used by the program
    - the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer that came down with the vertices
    - SEQ will not send vertex data until space in the register file has been allocated
4. SEQ sends the vector to the SP register file over the RE_SP interface (which has a bandwidth of 2048 bits/cycle)
    - the 64 vertex indices are sent to the 64 register files over 4 cycles
        - RF0 of SU0, SU1, SU2, and SU3 is written the first cycle
        - RF1 of SU0, SU1, SU2, and SU3 is written the second cycle
        - RF2 of SU0, SU1, SU2, and SU3 is written the third cycle
        - RF3 of SU0, SU1, SU2, and SU3 is written the fourth cycle
    - the index is written to the least significant 32 bits **(floating point format?) (what about compound indices)** of the 128-bit location within the register file (w); the remaining data bits are set to zero (x, y, z)
5. SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of ~~texture~~fetch state machine 0, or TSM0 FIFO)
    - the control packet contains the state pointer, the tag to the position cache and a register file base pointer.
6. TSM0 accepts the control packet and fetches the instructions for ~~texture~~fetch clause 0 from the global instruction store
    - TSM0 was first selected by the TSM arbiter before it could start
7. all instructions of ~~texture~~fetch clause 0 are issued by TSM0
8. the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)
    - TSM0 does not wait for requests made to the ~~Texture~~Fetch Unit to complete; it passes the register file write index for the ~~texture~~fetch data to the TU, which will write the data to the RF as it is received
    - once the TU has written all the data to the register files, it increments a counter that is associated with ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go ahead start to execute the ALU clause
9. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store
10. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of ~~texture~~fetch state machine 1, or TSM1 FIFO)
11. the control packet continues to travel down the path of reservation stations until all clauses have been executed
    - position can be exported in ALU clause 3 (or 4?); the data (and the tag) is sent over a position bus (which is shared with all four shader pipes) back to the PA's position cache
    - A parameter cache pointer is also sent along with the position data. This tells to the PA where the data is going to be in the parameter cache.
        - there is a position export FIFO in the SP that buffers position data before it gets sent back to the PA
        - the ASM arbiter will prevent a packet from starting an exporting clause if the position export FIFO is full
    - parameter data is exported in clause 7 (as well as position data if it was not exported earlier)
        - parameter data is sent to the Parameter Cache over a dedicated bus
        - the SEQ allocates storage in the Parameter Cache, and the SEQ deallocates that space when there is no longer a need for the parameters (it is told by the PA when using a token).
        - the ASM arbiter will prevent a packet from starting on ASM7 if the parameter cache (or the position buffer if position is being exported) is full

12. after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program

~~19.1.2~~20.1.2 *Sequencer Control of a Vector of Pixels*

Formatted: Bullets and Numbering

1. **As with vertex shader programs, pixel shaders are loaded into the global instruction store by the CP**
   - At this point it is assumed that the pixel program is loaded into the instruction store and thus ready to be read.
2. the RE's Pixel FIFO is loaded with the barycentric coordinates for pixel quads by the detailed walker
   - the state pointer and the LOD correction bits are also placed in the Pixel FIF0
   - the Pixel FIFO is wide enough to source four quad's worth of barycentrics per cycle
3. SEQ arbitrates between Pixel FIFO and Vertex FIFO – when there are no vertices pending OR there is no space left in the register files for vertices, the Pixel FIFO is selected
4. SEQ allocates space in the SP register file for all the GPRs used by the program
   - the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer
   - SEQ will not allow interpolated data to be sent to the shader until space in the register file has been allocated
5. SEQ controls the transfer of interpolated data to the SP register file over the RE_SP interface (which has a bandwidth of 2048 bits/cycle). See interpolated data bus diagrams for details.
6. SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of ~~texture~~fetch state machine 0, or TSM0 FIFO)
   - note that there is a separate set of reservation stations/arbiters/state machines for vertices and for pixels
   - the control packet contains the state pointer, the register file base pointer, and the LOD correction bits
   - all other informations (such as quad address for example) travels in a separate FIFO
7. TSM0 accepts the control packet and fetches the instructions for ~~texture~~fetch clause 0 from the global instruction store
   - TSM0 was first selected by the TSM arbiter before it could start
8. all instructions of ~~texture~~fetch clause 0 are issued by TSM0
9. the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)
   - TSM0 does not wait for ~~texture~~fetch requests made to the ~~Texture~~Fetch Unit to complete; it passes the register file write index for the ~~texture~~fetch data to the TU, which will write the data to the RF as it is received
   - once the TU has written all the data for a particular clause to the register files, it increments a counter that is associated with the ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go ahead and pop the FIFO and start to execute the ALU clause
10. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store
11. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of ~~texture~~fetch state machine 1, or TSM1 FIFO)
12. the control packet continues to travel down the path of reservation stations until all clauses have been executed
    - pixel data is exported in the last ALU clause (clause 7)
      - it is sent to an output FIFO where it will be picked up by the render backend
      - the ASM arbiter will prevent a packet from starting on ASM7 if the output FIFO is full
13. after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program

~~19.1.3~~20.1.3 *Notes*

Formatted: Bullets and Numbering

14. the state machines and arbitrers will operate ahead of time so that they will be able to immediately start the real threads or stall.

15. the register file base pointer for a vector needs to travel with the vector through the reservation stations, but the instruction store base pointer does not – this is because the RF pointer is different for all threads, but the IS pointer is only different for each state and thus can be accessed via the state pointer

16. Waterfalling, parameter buffer allocation, loops and branches and parameter cache de-allocation still needs to be specked out.

~~20.~~21. Open issues

Formatted: Bullets and Numbering

There is currently an issue with constants. If the constants are not the same for the whole vector of vertices, we don't have the bandwith from the ~~texture~~fetch store to feed the ALUs. Two solutions exists for this problem:

1) Let the compiler handle the case and put those instructions in a ~~texture~~fetch clause so we can use the bandwith there to operate. This requires a significant amount of temporary storage in the register store.
2) Waterfall down the pipe allowing only at a given time the vertices having the same constants to operate in parrallel. This might in the worst case slow us down by a factor of 16.

Need to do some testing on the size of the register file as well as on the register file allocation method (dynamic VS static).

Saving power?

Size of the fifo containing the information of a vector of pixels/vertices. And size of the fifos before the reservation stations.

Loops and branches.

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 2015~~19~~ October, 200117	GEN-CXXXXX-REVA	1 of 27

Author: Laurent Lefebvre

Issue To:	**Copy No:**

R400 Sequencer Specification

SEQ

Version 0.9~~8~~

Overview: This is an architectural specification for the R400 Sequencer block (SEQ). It provides an overview of the required capabilities and expected uses of the block. It also describes the block interfaces, internal sub-blocks, and provides internal state diagrams.

AUTOMATICALLY UPDATED FIELDS:

Document Location: C:\perforce\r400\arch\doc\gfx\RE\R400_Sequencer.doc

Current Intranet Search Title: R400 Sequencer Specification

APPROVALS	
Name/Dept	Signature/Date

Remarks:

THIS DOCUMENT CONTAINS CONFIDENTIAL INFORMATION THAT COULD BE SUBSTANTIALLY DETRIMENTAL TO THE INTEREST OF ATI TECHNOLOGIES INC. THROUGH UNAUTHORIZED USE OR DISCLOSURE.

Table Of Contents

1. **OVERVIEW** ....4~~3~~
1.1 Top Level Block Diagram...............................6~~5~~
1.2 Data Flow graph...8~~7~~
1.3 Control Graph ....9~~8~~
2. **INTERPOLATED DATA BUS** ....9~~8~~
3. **INSTRUCTION STORE**.............................12~~11~~
4. **SEQUENCER INSTRUCTIONS**.................12~~11~~
5. **CONSTANT STORE**..................................12~~11~~
6. **LOOPING AND BRANCHES**....................12~~11~~
6.1 The controlling state...................................12~~11~~
6.2 The Control Flow Program.........................13~~12~~
6.3 Data dependant predicate instructions.......16~~14~~
6.4 Register file indexing..................................17~~14~~
7. **REGISTER FILE ALLOCATION**................17~~15~~
8. **FETCH ARBITRATION**.........................18~~1616~~
9. **ALU ARBITRATION**..............................18~~1616~~
10. **HANDLING STALLS**.............................19~~1716~~
11. **CONTENT OF THE RESERVATION STATION FIFOS**19~~1716~~
12. **THE OUTPUT FILE**...............................19~~1716~~
13. **IJ FORMAT**..19~~1716~~
14. **THE PARAMETER CACHE**...................20~~1817~~
15. **VERTEX POSITION EXPORTING**.........20~~1817~~
16. **REAL TIME COMMANDS**.....................20~~1817~~
17. **REGISTERS**..21~~1818~~
18. **INTERFACES**..21~~1918~~
18.1 External Interfaces ....21~~1918~~
18.1.1 PA/SC to RE : IJ bus ....21~~1918~~
18.1.2 PA/SC to SEQ : IJ Control bus.22~~1918~~
18.1.3 VGT to RE : Vertex Bus ....22~~2019~~
18.1.4 VGT to SEQ : Vertex Control Bus 22~~2019~~
18.1.5 CP to SEQ : Constant store load 22~~2019~~
18.1.6 CP to SEQ : Fetch State store load 22~~2019~~
18.1.7 CP to SEQ : Control State store load 23~~2020~~
18.1.8 MH to SEQ: Instruction store Load 23~~2020~~
18.1.9 SP to RB : Pixel read from RBs23~~2020~~
18.1.10 SEQ to RB : Control bus ....23~~2020~~
18.1.11 RB to SEQ : Output file control.23~~2120~~
18.1.12 SP to RB : Position return bus..23~~2120~~
18.1.13 Shader Engine to Fetch Unit Bus (Fast Bus) 24~~2121~~
18.1.14 Sequencer to Fetch Unit bus (Slow Bus) 24~~2121~~
19. **INTERNAL INTERFACES**.....................24~~2221~~
20. **EXAMPLES OF PROGRAM EXECUTIONS** 24~~2221~~
20.1.1 Sequencer Control of a Vector of Vertices 24~~2221~~
20.1.2 Sequencer Control of a Vector of Pixels 26~~2322~~
20.1.3 Notes ....26~~2423~~
21. **OPEN ISSUES**......................................27~~2423~~

Revision Changes:

Rev 0.1 (Laurent Lefebvre) Date: May 7, 2001	First draft.
Rev 0.2 (Laurent Lefebvre) Date : July 9, 2001	Changed the interfaces to reflect the changes in the SP. Added some details in the arbitration section.
Rev 0.3 (Laurent Lefebvre) Date : August 6, 2001	Reviewed the Sequencer spec after the meeting on August 3, 2001.
Rev 0.4 (Laurent Lefebvre) Date : August 24, 2001	Added the dynamic allocation method for register file and an example (written in part by Vic) of the flow of pixels/vertices in the sequencer.
Rev 0.5 (Laurent Lefebvre) Date : September 7, 2001	Added timing diagrams (Vic)
Rev 0.6 (Laurent Lefebvre) Date : September 24, 2001	Changed the spec to reflect the new R400 architecture. Added interfaces.
Rev 0.7 (Laurent Lefebvre) Date : October 5, 2001	Added constant store management, instruction store management, control flow management and data dependant predication.
Rev 0.8 (Laurent Lefebvre) Date : October 8, 2001	Changed the control flow method to be more flexible. Also updated the external interfaces.
Rev 0.9 (Laurent Lefebvre) Date : October 17, 2001	Incorporated changes made in the 10/18/01 control flow meeting. Added a NOP instruction, removed

the conditional_execute_or_jump. Added debug registers.

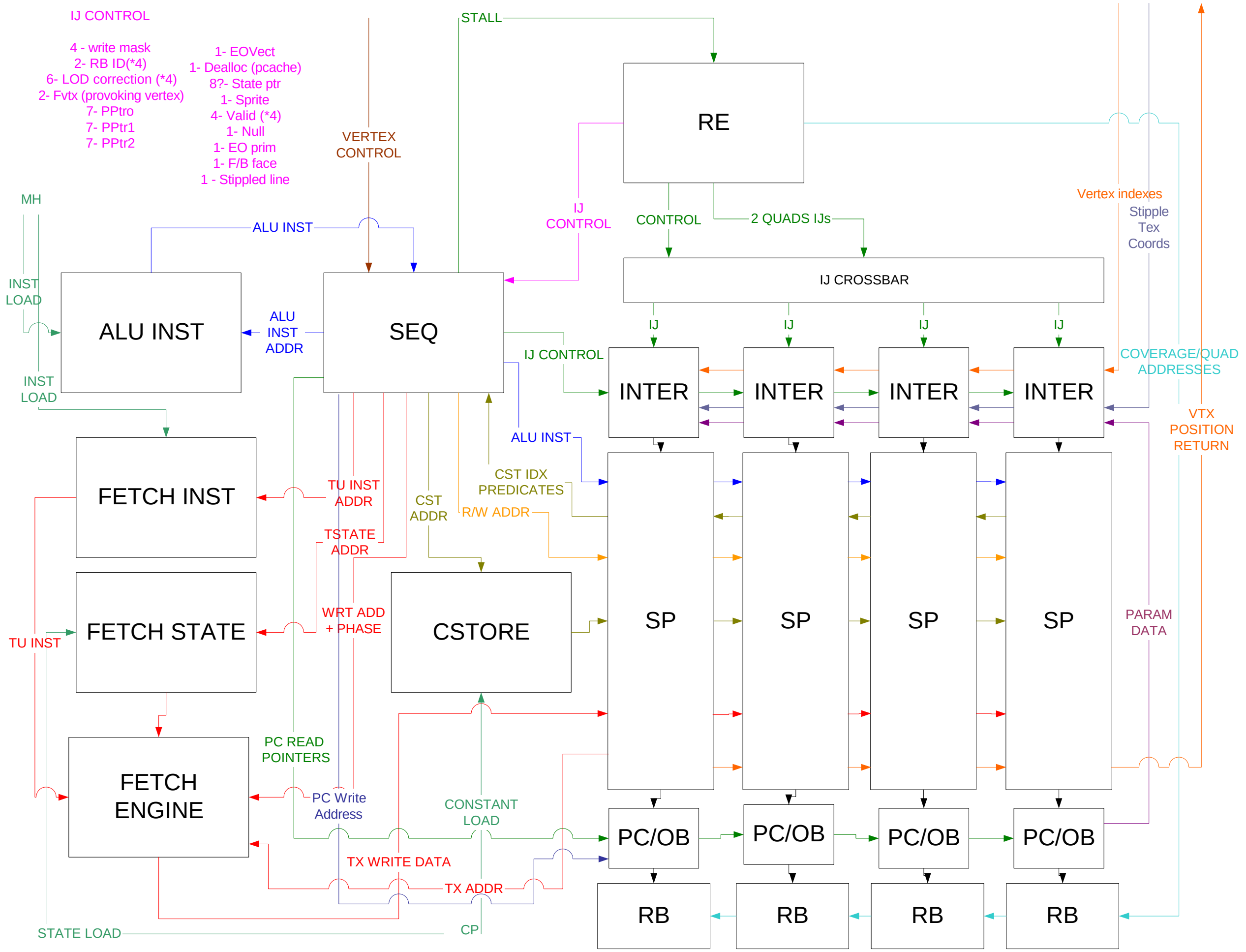
1. Overview

The sequencer first arbitrates between vectors of 64 vertices that arrive directly from primitive assembly and vectors of 16 quads (64 pixels) that are generated in the raster engine.

The vertex or pixel program specifies how many GPR's it needs to execute. The sequencer will not start the next vector until the needed space is available.

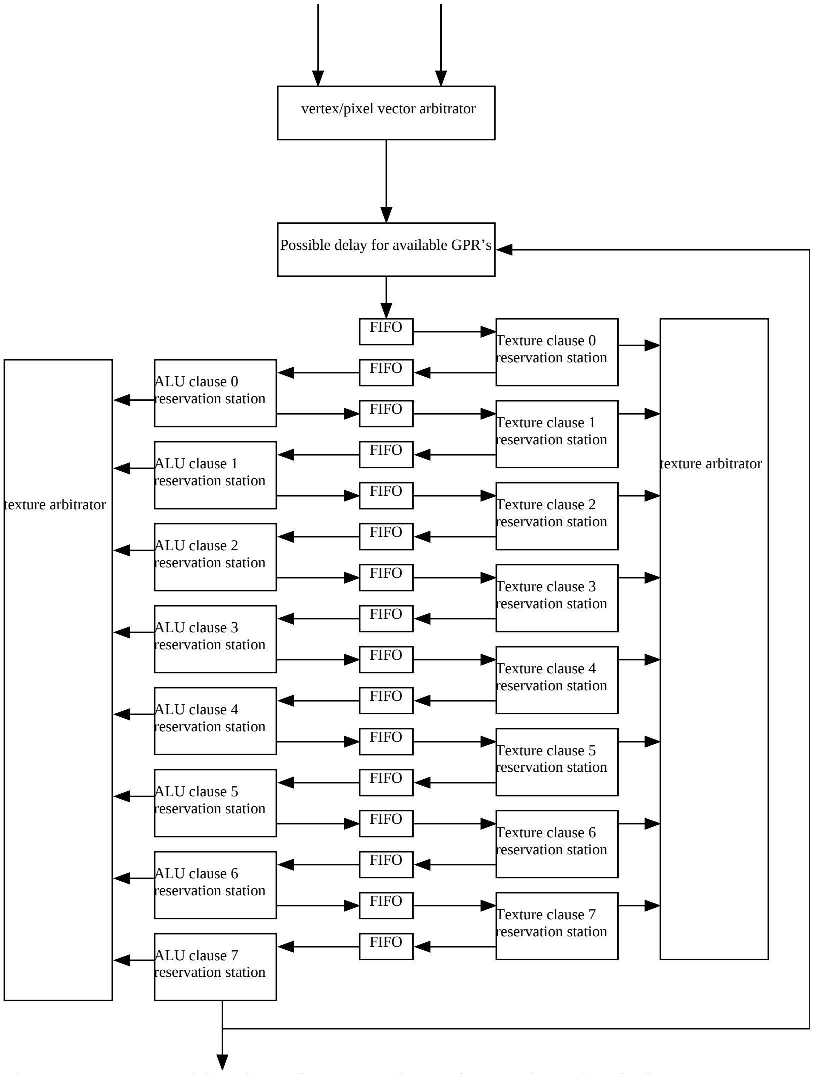
The sequencer is based on the R300 design. It chooses two ALU clauses and a fetch clause to execute, and executes all of the instructions in a clause before looking for a new clause of the same type. Two ALU clauses are executed interleaved to hide the ALU latency. Each vector will have eight fetch and eight ALU clauses, but clauses do not need to contain instructions. A vector of pixels or vertices ping-pongs along the sequencer FIFO, bouncing from fetch reservation station to alu reservation station. A FIFO exists between each reservation stage, holding up vectors until the vector currently occupying a reservation station has left. A vector at a reservation station can be chosen to execute. The sequencer looks at all eight alu reservation stations to choose an alu clause to execute and all eight fetch stations to choose a fetch clause to execute. The arbitrator will give priority to clauses/reservation stations closer to the bottom of the pipeline. It will not execute an alu clause until the fetch fetches initiated by the previous fetch clause have completed. There are two separate sets of reservation stations, one for pixel vectors and one for vertices vectors. This way a pixel can pass a vertex and a vertex can pass a pixel.

To support the shader pipe the raster engine also contains the shader instruction cache and constant store. There are only one constant store for the whole chip and one instruction store. These will be shared among the four shader pipes. The four shader pipes also execute the same instuction thus there is only one sequencer for the whole chip.



IJ CONTROL
4 - write mask
2- RB ID(*4)
6- LOD correction (*4)
2- Fvtx (provoking vertex)
7- PPtro
7- PPtr1
7- PPtr2
1- EOVect
1- Dealloc (pcache)
8?- State ptr
1- Sprite
4- Valid (*4)
1- Null
1- EO prim
1- F/B face
1 - Stippled line
VERTEX CONTROL
STALL
RE
IJ CONTROL
CONTROL
2 QUADS IJs
Vertex indexes
Stipple Tex Coords
MH
ALU INST
INST LOAD
ALU INST
ALU INST ADDR
SEQ
IJ CROSSBAR
IJ
IJ
IJ
IJ
IJ CONTROL
INTER
INTER
INTER
INTER
COVERAGE/QUAD ADDRESSES
INST LOAD
VTX POSITION RETURN
ALU INST
FETCH INST
TU INST ADDR
CST ADDR
CST IDX PREDICATES
R/W ADDR
TSTATE ADDR
SP
SP
SP
SP
PARAM DATA
TU INST
FETCH STATE
WRT ADD + PHASE
CSTORE
PC READ POINTERS
FETCH ENGINE
PC Write Address
CONSTANT LOAD
PC/OB
PC/OB
PC/OB
PC/OB
TX WRITE DATA
TX ADDR
RB
RB
RB
RB
STATE LOAD
CP

1.1 Top Level Block Diagram



vertex/pixel vector arbitrator
Possible delay for available GPR's
FIFO
Texture clause 0 reservation station
Texture clause 1 reservation station
Texture clause 2 reservation station
Texture clause 3 reservation station
Texture clause 4 reservation station
Texture clause 5 reservation station
Texture clause 6 reservation station
Texture clause 7 reservation station
texture arbitrator
ALU clause 0 reservation station
ALU clause 1 reservation station
ALU clause 2 reservation station
ALU clause 3 reservation station
ALU clause 4 reservation station
ALU clause 5 reservation station
ALU clause 6 reservation station
ALU clause 7 reservation station
texture arbitrator

There are two sets of the above figure, one for vertices and one for pixels.

The rasterizer always checks the vertices FIFO first and if allowed by the sequencer sends the data to the shader. If the vertex FIFO is empty then, the rasterizer takes the first entry of the pixel FIFO (a vector of 64 pixels) and sends it to the interpolators. Then the sequencer takes control of the packet. The packet consists of 21 bits of state, 6-7 bits for the base address of the Shader program and some information on the coverage to determine fetch LOD plus other various small state bits. ~~All other information (2x2 adresses) is put in a FIFO (one for the pixels and one for the vertices) and retrieved when the packet finishes its last clause.~~

On receipt of a packet, the input state machine (not pictured but just before the first FIFO) allocated enough space in the registers to store the interpolated values and temporaries. Following this, the input state machine stacks the packet in the first FIFO.

On receipt of a command, the level 0 fetch machine issues a texure request and corresponding register address for the fetch address (ta). A small command (tcmd) is passed to the fetch system identifying the current level number (0) as well as the register write address for the fetch return data. One fetch request is sent every 4 clocks causing the texturing of sixteen 2x2s worth of data (or 64 vertices). Once all the requests are sent the packet is put in FIFO 1.

Upon recept of the return data, the fetch unit writes the data to the register file using the write address that was provided by the level 0 fetch machine and sends the clause number (0) to the level 0 fetch state machine to signify that the write is done and thus the data is ready. Then, the level 0 fetch machine increments the counter of FIFO 1 to signify to the ALU 1 that the data is ready to be processed.

On receipt of a command, the level 0 ALU machine first decrements the input FIFO counter and then issues a complete set of level 0 shader instructions. For each instruction, the state machine generates 3 source addresses, one destination address (3 cycles later) and an instruction. Once the last instruction as been issued, the packet is put into FIFO 2.

There will always be two active ALU clauses at any given time (and two arbitrers). One arbitrer will arbitrate over the odd instructions (4 clocks cycles) and the other one will arbitrate over the even instructions (4 clocks cycles). The only constraints between the two arbitrers is that they are not allowed to pick the same clause number as they other one is currently working on if the packet os of the same type.

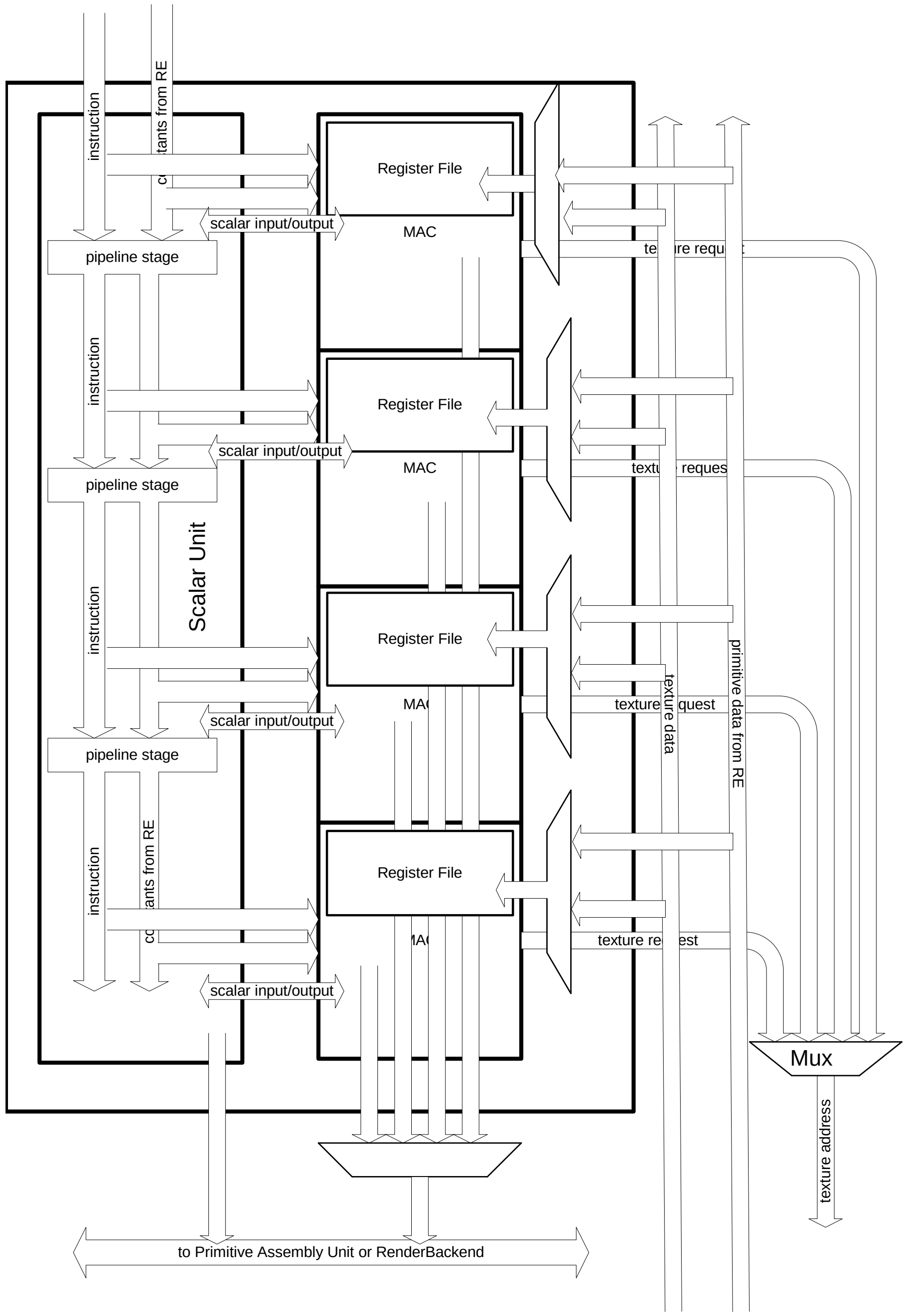
If the packet is a vertex packet, upon reaching ALU clause 3~~4~~, it can export the position if the position is ready. So the arbitrer must prevent ALU clause 3~~4~~ to be selected if the positional buffer is full (or can't be accessed). Along with the positional data, the location where the vertex data is to be put is also sent (parameter data pointers).

A special case is for HOS surfaces wich can export 12 parameters per clause to the output buffer. If the output buffer is full or doesn't have enough space the sequencer will prevent such a vertex group to enter an exporting clause.

All other level process in the same way until the packet finally reaches the last ALU machine (8). On completion of the level 8 ALU clause, a valid bit is sent to the Render Backend which picks up the color data. This requires that the last instruction writes to the output register – a condition that is almost always true. If the packet was a vertex packet, instead of sending the valid bit to the RB, it is sent to the PA so it can know that the data present in the parameter store is valid.

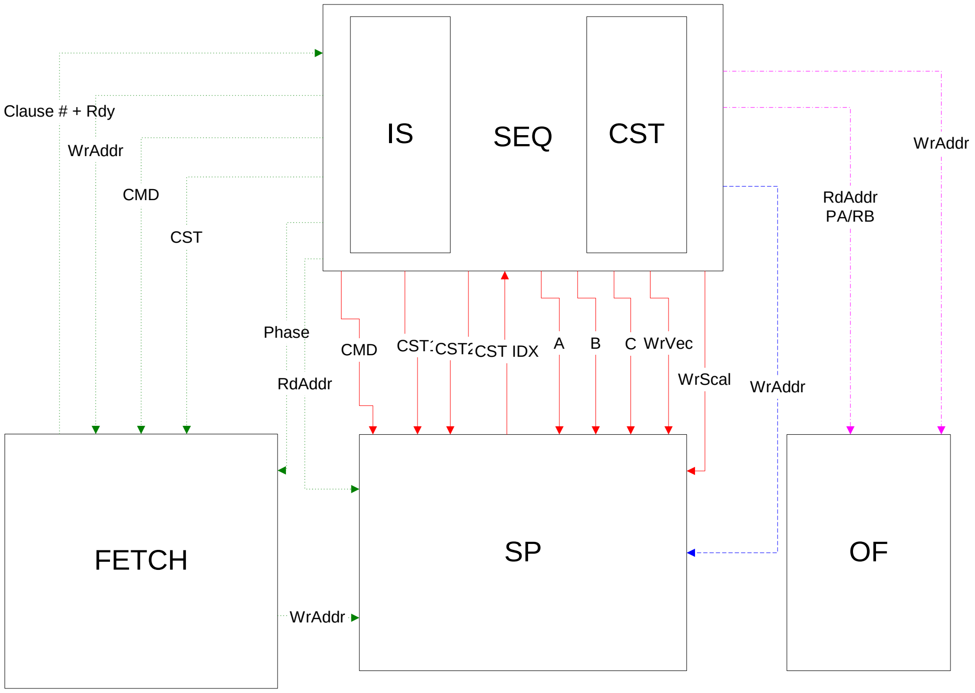
Only two ALU state machine may have access to the register file address bus or the instruction decode bus at one time. Similarly, only one fetch state machine may have access to the register file address bus at one time. Arbitration is performed by three arbitrer blocks (two for the ALU state machines and one for the fetch state machines). The arbitrers always favor the higher number state machines, preventing a bunch of half finished jobs from clogging up the register files.

1.2 Data Flow graph



The gray area represents blocks that are replicated 4 times per shader pipe (16 times on the overall chip).

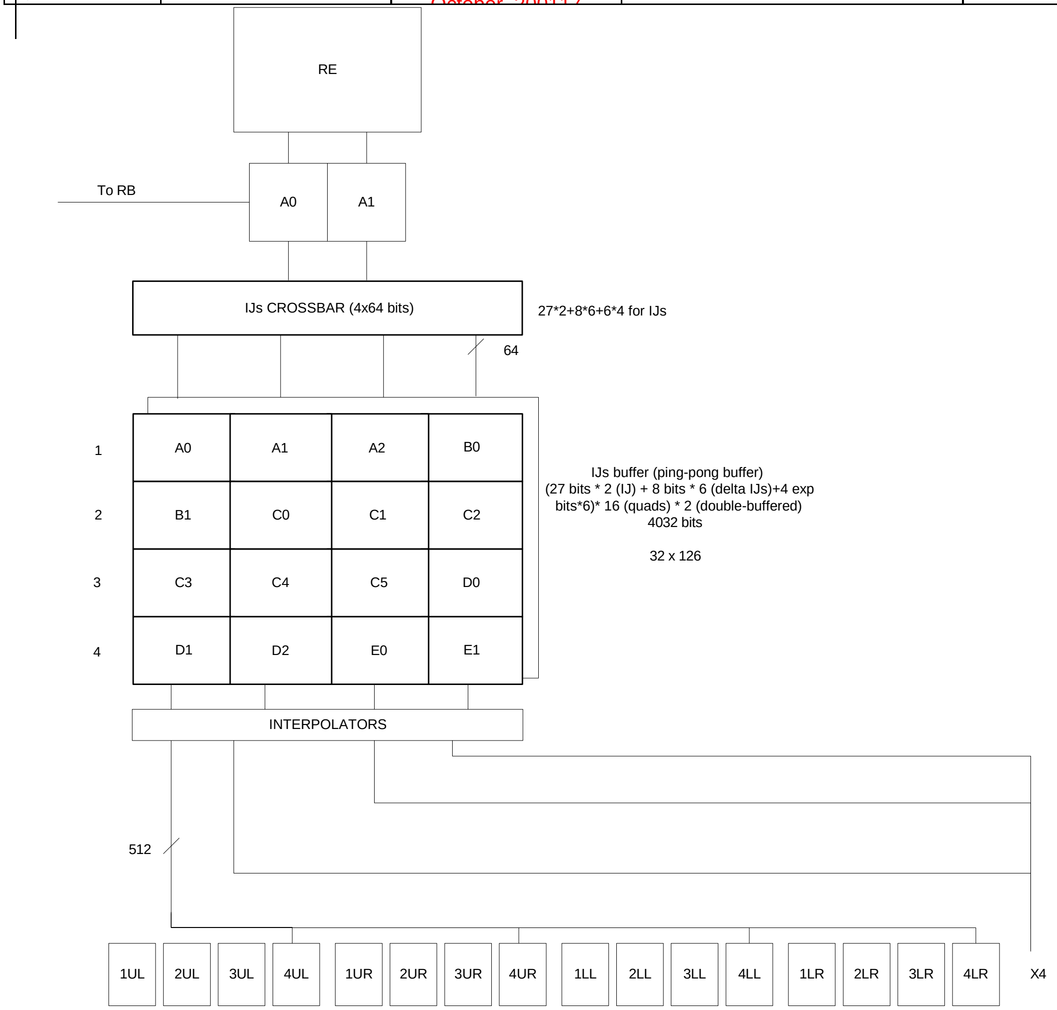
1.3 Control Graph



In green is represented the Fetch control interface, in red the ALU control interface, in blue the Interpolated/Vector control interface and in purple is the output file control interface.

2. Interpolated data bus

The interpolators contain an IJ buffer to pack the information as much as possible before writing it to the register file.



RE
To RB
A0
A1
IJs CROSSBAR (4x64 bits)
27*2+8*6+6*4 for IJs
64
1
2
3
4
A0
A1
A2
B0
B1
C0
C1
C2
C3
C4
C5
D0
D1
D2
E0
E1
IJs buffer (ping-pong buffer)
(27 bits * 2 (IJ) + 8 bits * 6 (delta IJs)+4 exp bits*6)* 16 (quads) * 2 (double-buffered)
4032 bits
32 x 126
INTERPOLATORS
512
1UL
2UL
3UL
4UL
1UR
2UR
3UR
4UR
1LL
2LL
3LL
4LL
1LR
2LR
3LR
4LR
X4

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19
SP0	A0	B1	C3	D1					A0	B1	C3	D1					V 0-3	V 16-19	V 32-35	V 48-51
SP1	A1		C4	D2		C0			A1		C4	D2		C0			V 4-7	V 20-23	V 36-39	V 52-55
SP2	A2		C5			C1		E0	A2		C5			C1		E0	V 8-11	V 24-27	V 40-43	V 56-59
SP3					B0	C2	D0	E1					B0	C2	D0	E1	V 12-15	V 28-31	V 44-47	V 60-63

P0

P1

Above is an example of a tile we might receive. The IJ information is packed in the IJ buffer 2 quads at a time. The sequencer allows at any given time as many as four quads to interpolate a parameter. They all have to come from the same primitive. Then the sequencer controls the write mask to the register to write the valid data in.

3. Instruction Store

There is going to be only one instruction store for the whole chip. It will contain 4096 instructions of 96 bits each. There is also going to be a control instruction store of size 256(512?)x32.

{ISSUE : The instruction store is loaded by the sequencer using the memory hub ?}.

The read bandwith from this store is 96*2 bits/ 4 clocks (48 bits/clock). It is likely to be a 1 port memory; we use 1 clock to load the ALU instruction, 1 clocks to load the Fetch instruction, 1 clock to load 2 control flow instructions and 1 clock to write instructions.

4. Sequencer Instructions

All control flow instructions and move instructions are handled by the sequencer only. The ALUs will perform NOPs during this time (MOV PV,PV, PS,PS).

5. Constant Store

The constant store is managed by the CP. The sequencer is aware of where the constants are using a remaping table also managed by the CP. A likely size for the constant store is 512x128 bits. The constant store is also planned to be shared. The read BW from the constant store is 128 bits/clock and the write bandwith is 32/4 bits/clock.

In order to do constant store indexing, the sequencer must be loaded first with the indexes (that come from the GPRs). There are 144 wires from the exit of the SP to the sequencer (9 bits pointers x 16 vertexes/clock). Since the data must pass thru the Shader pipe for the float to fixed convertion, there is a latency of 4 clocks (1 instruction) between the time the sequencer is loaded and the time one can index into the constant store. The assembly will look like this

```
MOVA  R1.X,R2.X       // Loads the sequencer with the content of R2.X, also copies the content of R2.X into R1.X
NOP                   // latency of the float to fixed conversion
ADD   R3,R4,C0[R2.X]// Uses the state from the sequencer to add R4 to C0[R2.X] into R3
```

Note that we don't really care about what is in the brackets because we use the state from the MOVA instruction. R2.X is just written again for the sake of simplicity.

The storage needed in the sequencer in order to support this feature is 2*64*9 bits = 1152 bits.

6. Looping and Branches

Loops and branches are planned to be supported and will have to be dealt with at the sequencer level. We plan on supporting constant loops and branches using a control program. ~~The control program has 4(5) instructions:~~

6.1 The controlling state.

As per Dx9 the following state is available for control flow:

Boolean[15:0]
loop_count[7:0][7:0]
In addition:
loop_start [7:0] [7:0]
loop_step [7:0] [7:0]
Exist to give more control to the controlling program.

We will extend that in the R400 to:

Boolean[255:0]
Loop_count[7:0][15:0]
Loop_Start[7:0] [15:0] times 2 (one for constant,registert)
Loop_Step[7:0] [15:0] times 2 (one for constant,register)
Loop_End[7:0] [15:0]

{ISSUE: How is the controlling state loaded and how many contexts do we have?}

We have a stack of 4 elements for calling subroutines and 4 loop counters to allow for nested loops.

We also keep 8 predicate vectors and 8 AND/OR sets of 3 bits. These bits can be 0: all 0s, 1: all ones and 11: mixed.

6.2 The Control Flow Program

The R300 uses a match method for control flow: The shader is executed, and at every instruction its address is compared with addresses (or address?) in a control table. The “event” in the control table can redirect operations in the program.

The Method chosen for the R400 is a “control program”. The control program has ten basic instructions:

Execute
Conditional_execute
Conditional_Execute_Predicates
Conditional_execute_or_Jump
Conditional_jump
Call
Return
Loop_start
Loop_end
End_of_clause

Execute, causes the specified number of instructions in instruction store to be executed.
Conditional_execute checks a condition first, and if true, causes the specified number of instructions in instruction store to be executed.
Loop_start resets the corresponding loop counter to the start value on the first pass after it checks for the end condition and if met jumps over to a specified address.
Loop_end increments (decrements?) the loop counter and jumps back the specified number of instructions.
Call jumps to an address and pushes the IP counter on the stack. On the return instruction, the IP is poped from the stack.
Conditional_execute_or_Jump executes a block of instructions or jumps to an address is the condition is not met.
Conditional_execute_Predicates executes a block of instructions if all bits in the predicate vectors meet the condition.
End_of_clause marks the end of a clause.
Conditional_jumps jumps to an address if the condition is met.
NOP is a regular NOP

NOTE THAT ALL JUMPS MUST JUMP TO EVEN CFP ADDRESSES. Thus the compiler must insert NOPs where needed to align the jumps on even CFP addresses.

Also if the jump is logically bigger than 4096 we break the program and set the debug registers.

We have to fit instructions into 48 bits in order to be able to put two control flow instruction per line in the instruction store.

Execute				
47	46… 42	41 … 24	23 … 12	11 … 0
Addressing	00001	RESERVED	Instruction _count	Exec Address

Execute up to 4k instructions at the specified address in the instruction memory.

~~Conditionnal_Execute_or_Jump~~NOP		
47	46 … 42	~~41 … 34~~ ~~33~~ ~~32 … 21~~ ~~20 … 12~~ ~~11 … 0~~41 … 0
Addressing	00010	~~Booleans~~ ~~Condition~~ ~~Jump address~~ ~~Instruction_count~~ ~~Exec Address~~RESERVED

If the specified boolean (8 bits can address 256 booleans) meets the specified condition then execute the specified instructions (up to 512 instructions) or if the condition is not met jump to the jump address in the control flow program. This MUST be a forward jump.

Conditionnal_Execute						
47	46 … 42	41 … 34	33	32 … 24~~1~~	2~~0~~3 … 12	11 … 0
Addressing	00011	Boolean address	Condition	RESERVED	Instruction_count	Exec Address

If the specified boolean (8 bits can address 256 booleans) meets the specified condition then execute the specified instructions (up to 4k~~512~~ instructions)

Conditionnal_Execute_Predicates						
47	46 … 42	41 … 38	37	36 … 24~~1~~	2~~0~~3 … 12	11 … 0
Addressing	00100	Predicate vector	Condition	RESERVED	Instruction_count	Exec Address

Check the AND/OR of all current predicate bits. If AND/OR matches the condition execute the specified number of instructions.

Loop_Start				
47	46 … 42	41 … 16	15 … 4	3 … 0
Addressing	00101	RESERVED	Jump address	Loop ID

Loop Start. Compares the loop count with the end value. If loop condition not met jump to the address. Forward jump only. Also computes the index value.

Loop_End				
47	46 … 42	41 … 16	15 … 4	3 … 0
Addressing	00111	RESERVED	Start address	Loop ID

Loop end. Increments the counter by one and jumps BACK only to the start of the loop.

The way this is described does not prevent nested loops, and the inclusion of the loop id make this easy to do.

Call			
47	46 … 42	41…12	11 … 0
Addressing	01000	RESERVED	Address

Jumps to the specified address and pushes the IP counter on the stack.

Return		
47	46 … 42	41 … 0
Addressing	01001	RESERVED

Pops the topmost address from the stack and jumps to that address. If nothing is on the stack, the program will just continue to the next instruction.

Conditionnal_Jump					
47	46 … 42	41 … 34	33	32 … 12	11 … 0
Addressing	01010	Boolean address	Condition	RESERVED	Address

If condition met, jumps to the address. FORWARD jump only allowed.

End_of_Clause		
47	46 … 42	41 … 0
Addressing	01011	RESERVED

Marks the end of a clause.

To prevent infinite loops, we will keep 9 bits loop counters instead of 8 (we are only able to loop 256 times). If the counter goes higher than 255 then the loop_end or the loop_start is going to break the loop and set de debug registers. The sequencer will keep ~~a~~ two loop indexes values~~ of 17 bits~~:

IC index for constant indexing (9 bits)

IR index for register file indexing (7 bits)

~~.~~ This will be updated everytime we loop and can only be used to index the constant store and the register file. The way to compute this value is:

Index = Loop_counter*Loop_iterator + Loop_init.

The IC for constant is going to return 0 if it is out of the constant range. The IR index is going to break the program if the index exeeds the number of requested registers.

The basic model is as follows:

The render state defined the clause boundaries:
Vertex_shader_fetch[7:0][7:0] // eight 8 bit pointers to the location where each clauses control program is located
Vertex_shader_alu[7:0][7:0] // eight 8 bit pointers to the location where each clauses control program is located
Pixel_shader_fetch[7:0][7:0] // eight 8 bit pointers to the location where each clauses control program is located
Pixel_shader_alu[7:0][7:0] // eight 8 bit pointers to the location where each clauses control program is located

The control program for a given clause is executed to completion before moving to another clause, (with the exception of the pick two nature of the alu execution). The control program is the only program aware of the clause boundaries.

6.3 Data dependant predicate instructions

Data dependant conditionals will be supported in the R400. The only way we plan to support those is by supporting three vector/scalar predicate operations of the form:

PRED_SETE_# - similar to SETE except that the result is 'exported' to the sequencer.
PRED_SETGT_# - similar to SETGT except that the result is 'exported' to the sequencer
PRED_SETGTE_# - similar to SETGTE except that the result is 'exported' to the sequencer

For the scalar operations only we will also support the two following instructions:
PRED_SETE0_# – SETE0
PRED_SETE1_# – SETE1

The export is a single bit - 1 or 0 that is sent using the same data path as the MOVA instruction. The sequencer will maintain 4 sets of 64 bit predicate vectors (in fact 8 sets because we interleave two programs but only 4 will be exposed) and use it to control the write masking. This predicate is not maintained across clause boundaries. The # sign is used to specify wich predicate set you want to use 0 thru 3.

Then we have two conditional execute bits. The first bit is a conditional execute “on” bit and the second bit tells us if we execute on 1 or 0. For exemple, the instruction :

P0_ADD_# R0,R1,R2

Is only going to write the result of the ADD into those GPRs whose predicate bit is 0. Alternatively, P1_ADD_# would only write the results to the GPRs whose predicate bit is set. The use of the P0 or P1 without precharging the sequencer with a PRED instruction is undefined.

{Issue: do we have to have a NOP between PRED and the first instruction that uses a predicate?}

6.4 Register file indexing

Because we can have loops in fetch clause, we need to be able to index into the register file in order to retrieve the data created in a fetch clause loop and use it into an ALU clause. The instruction will include the base address for register indexing and the instruction will contain these controls :

Bit7	Bit 6	
0	0	'absolute register'
0	1	'relative register'
1	0	'previous vector'
1	1	'previous scalar'

In the case of an absolute register we just take the address as is. In the case of a relative register read we take the base address and we add to it the loop_index and this becomes our new address that we give to the shader pipe.

Formatted: Bullets and Numbering

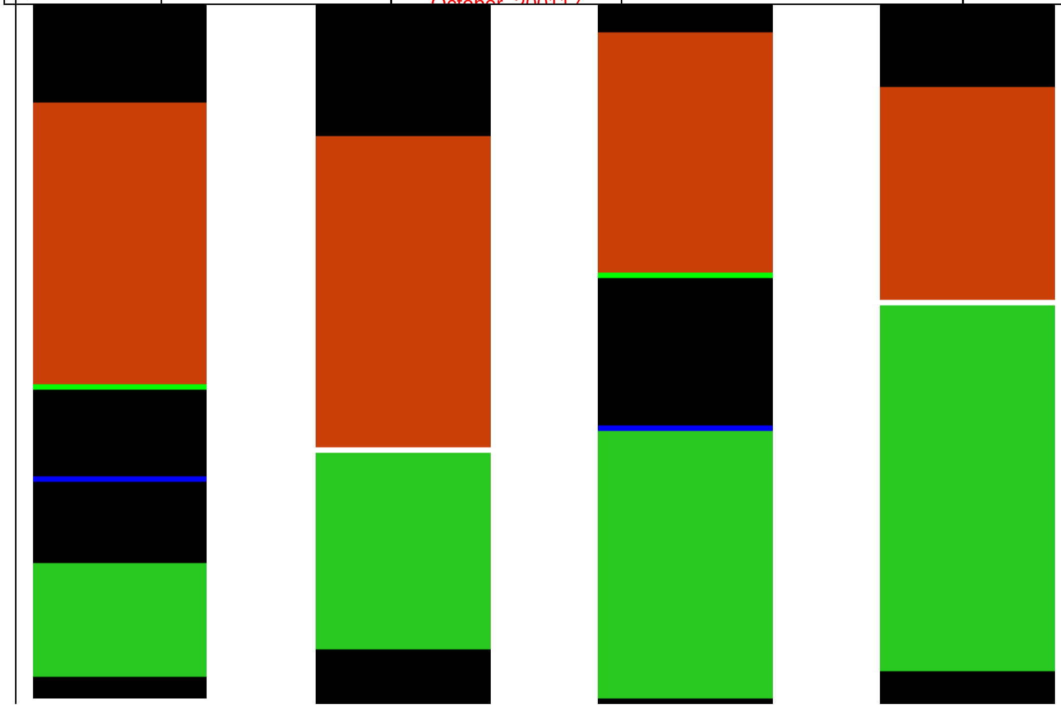
7. HOS surfaces

HOS surfaces are able to export from any clause but to memory ONLY. If they want to export to the parameter cache they have to do it in the last clause (7). They can also export position in clause 3. The buffer they want to export into must be specified in the "exports" field of the state registers.

Formatted: Bullets and Numbering

~~7.~~8. Register file allocation

The register file allocation for vertices and pixels can either be static or dynamic. In both cases, the register file in managed using two round robins (one for pixels and one for vertices). In the dynamic case the boundary between pixels and vertices is allowed to move, in the static case it is fixed to VERTEX_REG_SIZE for vertices and 256-VERTEX_REG_SIZE for pixels.



Above is an example of how the algorithm works. Vertices come in from top to bottom; pixels come in from bottom to top. Vertices are in orange and pixels in green. The blue line is the tail of the vertices and the green line is the tail of the pixels. Thus anything between the two lines is shared. When pixels meets vertices the line turns white and the boundary is static until both vertices and pixels share the same “unallocated bubble”. Then the boundary is allowed to move again.

~~8.~~9. Fetch Arbitration

Formatted: Bullets and Numbering

The fetch arbitration logic chooses one of the 8 potentially pending fetch clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute. Once chosen, the clause state machine will send one 2x2 fetch per clock (or 4 fetches in one clock every 4 clocks) until all the fetch instructions of the clause are sent. This means that there cannot be any dependencies between two fetches of the same clause.

The arbitrator will not wait for the fetches to return prior to selecting another clause for execution. The fetch pipe will be able to handle up to X(?) in flight fetches and thus there can be a fair number of active clauses waiting for their fetch return data.

~~9.~~10. ALU Arbitration

Formatted: Bullets and Numbering

ALU arbitration proceeds in almost the same way than fetch arbitration. The ALU arbitration logic chooses one of the 8 potentially pending ALU clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute. There are two ALU arbitrers, one for the even clocks and one for the odd clocks. For exemple, here is the sequencing of two interleaved ALU clauses (E and O stands for Even and Odd sets of 4 clocks):

Einst0 Oinst0 Einst1 Oinst1 Einst2 Oinst2 Einst0 Oinst3 Einst1 Oinst4 Einst2 Oinst0…
Proceeding this way hides the latency of 8 clocks of the ALUs.

Formatted: Bullets and Numbering

~~10.~~11. Handling Stalls

When the output file is full, the sequencer prevents the ALU arbitration logic to select the last clause (this way nothing can exit the shader pipe until there is place in the output file. If the packet is a vertex packet and the position buffer is full (POS_FULL) then the sequencer also prevents a thread to enter the exporting clause (4?). The sequencer will set the OUT_FILE_FULL signal n clocks before the output file is actually full and thus the ALU arbitrer will be able read this signal and act accordingly by not preventing exporting clauses to proceed.

Formatted: Bullets and Numbering

~~11.~~12. Content of the reservation station FIFOs

~~3~~21 bits of Render State ~~6~~77 bits for the base address of the ~~instruction~~ GPRs~~store~~, some bits for LOD correction and coverage mask information in order to fetch fetch for only valid pixels, quad address and 1 bit to specify if the vector is of pixels or vertices. ~~Every other information (such as the coverage mask, quad address, etc.) is put in a FIFO and is retrieved when the quad exits the shader pipe to enter in the output file buffer.~~ Since pixels and vertices are kept in order in the shader pipe, we only need two fifos (one for vertices and one for pixels) deep enough to cover the shader pipe latency. This size will be determined later when we will know the size of the small fifos between the reservation stations.

For texture clauses, 3 bits * 4 are going to be kept. These are the AND/OR of the predicate vectors. 0 for all 0s, 1 for all ones and MIXED.

Formatted: Bullets and Numbering

~~12.~~13. The Output File

The output file is where pixels are put before they go to the RBs. The write BW to this store is 256 bits/clock. Just before this output file are staging registers with write BW 512 bits/clock and read BW 256 bits/clock. For this reason only ONE concurrent program can be of clause 8 (exporting clause) the other program MUST not. The staging registers are 4x128 (and there are 16 of those on the whole chip).

Formatted: Bullets and Numbering

~~13.~~14. IJ Format

The IJ information sent by the PA is of this format on a per quad basis:

We have a vector of IJ's (one IJ per pixel at the centroid of the fragment or at the center of the pixel depending on the mode bit). The interpolation is done at a different precision across the 2x2. The upper left pixel's parameters are always interpolated at full 19x24 mantissa precision. Then the result of the interpolation along with the difference in IJ in reduced precision is used to interpolate the parameter for the other three pixels of the 2x2. Here is how we do it:

Assuming P0 is the interpolated parameter at Pixel 0 having the barycentric coordinates I(0), J(0) and so on for P1,P2 and P3. Also assuming that A is the parameter value at V0 (interpolated with I), B is the parameter value at V1 (interpolated with J) and C is the parameter value at V2 (interpolated with (1-I-J).

$$\Delta 01I = I(1) - I(0)$$
$$\Delta 01J = J(1) - J(0)$$
$$\Delta 02I = I(2) - I(0)$$
$$\Delta 02J = J(2) - J(0)$$
$$\Delta 03I = I(3) - I(0)$$
$$\Delta 03J = J(3) - J(0)$$

P0	P1
P2	P3

$$P0 = C + I(0) * (A - C) + J(0) * (B - C)$$
$$P1 = P0 + \Delta 01I * (A - C) + \Delta 01J * (B - C)$$
$$P2 = P0 + \Delta 02I * (A - C) + \Delta 02J * (B - C)$$
$$P3 = P0 + \Delta 03I * (A - C) + \Delta 03J * (B - C)$$

P0 is computed at 19x24 mantissa precision and P1 to P3 are computed at 8X24 mantissa precision. So far no visual degradation of the image was seen using this scheme.

Multiplies (Full Precision): 2
Multiplies (Reduced precision): 6
Subtracts 19x24 (Parameters): 2
Adds: 8

FORMAT OF P0's IJ : Mantissa 19 Exp 4 for I + Sign
Mantissa 19 Exp 4 for J + Sign

FORMAT of Deltas (x3):Mantissa 8 Exp 4 for I + Sign
Mantissa 8 Exp 4 for J + Sign

Total number of bits : 19*2 + 8*6 + 4*8 + 4*2 = 126

Formatted: Bullets and Numbering

~~14.~~15. The parameter cache

The parameter cache is where the vertex shaders export their data. It consists of 16 128x128 memories (1R/1W). The reuse engine will make it so that all vertexes of a given primitive will hit different memories.

Formatted: Bullets and Numbering

~~15.~~16. Vertex position exporting

On clause 4 (or 5) the vertex shader can export to the PA both the vertex position and the point sprite. It can also do so at clause 8 if not done at clause 4. Along with the position is exported a pointer to the parameter cache where the data will be once the vertex shader exports. The storage needed to perform the position export is at least 64x128 memories for the position and 64x32 memories for the sprite size. It is going to be taken in the pixel output fifo.

Formatted: Bullets and Numbering

~~16.~~17. Real time commands

We are unable to use the parameter memory since there is no way for a command stream to write into it. Instead we need to add three 16x128 memories (one for each of three vertices x 16 interpolants). These will be mapped onto the register bus and written by type 0 packets, and output to the the parameter busses (the sequencer and/or PA need to be able to address the realtime parameter memory as well as the regular parameter store. For higher performance we should be able able to view them as two banks of 16 and do double buffering allowing one to be loaded, while the other is rasterized with. Most overlay shaders will need 2 or 4 scalar coordinates, one option might be to restrict the memory to 16x64 or 32x64 allowing only two interpolated scalars per cycle, the only problem I see with this is, if we view support for 16 vector-4 interpolants important (true only if we map microsoft's high priority stream to the realtime stream), then the PA/sequencer need to support a realtime-specific mode where we need to address 32 vectors of parameters instead of 16.

~~17.~~18. Registers

Formatted: Bullets and Numbering

DYNAMIC_REG	Dynamic allocation (pixel/vertex) of the register file on or off.
VERTEX_REG_SIZE	What portion of the register file is reserved for vertices (static allocation only)
PIXEL_MIN_SIZE	Minimal size of the register file's pixel portion (dynamic only)
VERTEX_MIN_SIZE	Minimal size of the register file's vertex portion (dynamic only)
Vshader_fetch[11:0][7:0]	eight 12 bit pointers to the location where each clauses control program is located
Vshader_alu[11:0][7:0]	eight 12 bit pointers to the location where each clauses control program is located
Pshader_fetch[11:0][7:0]	eight 12 bit pointers to the location where each clauses control program is located
Pshader_alu[11:0][7:0]	eight 12 bit pointers to the location where each clauses control program is located
PSHADER	base pointer for the pixel shader
VSHADER	base pointer for the vertex shader
VWRAP	wrap point for the vertex shader instruction store
PWRAP	wrap point for the pixel shader instruction store
REG_ALLOC_PIX	number of registers to allocate for pixel shader programs
REG_ALLOC_VERT	number of registers to allocate for vertex shader programs
PARAM_MASK[0…16]	parameter mask to specify how parameters maps in the pixel shader
FLAT_GOUR[0…16]	wich parameters are to be gouraud shaded
GEN_TEX[0….16]	for wich parameters do we need to generate tex coords.
CYL_WRAP[0…64]	for wich parameters (and channels (xyzw)) do we do the cyl wrapping.
P_EXPORT[8]	number of exports for pixel shader
V_EXPORT[8]	number of exports for vertex shader for each clause. All numbers relate to the output buffer exports but for V_EXPORT[7] than can relate to the PC if Exports[7] is set to 00000.~~(also the number of interpolated parameters for pixel shaders)~~
~~V_EXPORT_LOC~~	~~Vertex shader exporting to RB or the PCACHE~~
ARBITRATION_policy	policy of the arbitration between vetexes and pixels
Exports[8][6]	Wich clause is exporting to the output buffer and what is it exporting.
	000000 : Not exporting (or exporting only to the PC)
	000001 : Exporting position (1)
	000010 : Exporting position (2)
	(1)00100 : Exporting RG
	(1)01000 : Exporting BA
	(1)10000 : Exporting Z
	If MSB set pixel shader exporting linear to memory not to Frame Buffer.
CST_SIZE_P	Size of the constant store for pixels
CST_SIZE_V	Size of the constant store for vertexes

19. DEBUG registers

Formatted: Bullets and Numbering

PROB_ADDR	instruction address where the first problem occurred
PROB_COUNT	number of problems encountered during the execution of the program

~~18.~~20. Interfaces

Formatted: Bullets and Numbering

~~18.1~~20.1 External Interfaces

~~18.1.1~~20.1.1 *PA/SC to RE : IJ bus*

This is a bus that sends the IJ information to the IJ fifos on the top of each shader pipe. At the same time the control information goes to the sequencer

Name	Direction	Bits	Description
IJs	PA→RE	63	IJ information sent over 2 clocks
Mask	PA→RE	1	Write Mask

~~18.1.2~~20.1.2 PA/SC to SEQ : IJ Control bus

Formatted: Bullets and Numbering

This is the control information sent to the sequencer in order to control the IJ fifos and all other information needed to execute a shader program on the sent pixels.

Name	Direction	Bits	Description
Write Mask	PA→SEQ(RE)	4	Quad Write mask left to right
RB_ID	PA→SEQ(RE)	8	RB id for each quad sent 2 bits per quad
LOD_CORRECT	PA→SEQ(RE)	24	LOD correction per quad (6 bits per quad)
FVTX	PA→SEQ(RE)	2	Provoking vertex for flat shading
PPTR0	PA→SEQ(RE)	11	P Store pointer for vertex 0
PPRT1	PA→SEQ(RE)	11	P Store pointer for vertex 1
PPTR2	PA→SEQ(RE)	11	P Store pointer for vertex 2
E_OFF_VECTOR	PA→SEQ(RE)	1	End of the vector
DEALLOC	PA→SEQ(RE)	1	Deallocation token for the P Store
STATE	PA→SEQ(RE)	21	State/constant pointer (6*3+3)
VALID	PA→SEQ(RE)	16	Valid bits for all pixels
NULL	PA→SEQ(RE)	1	Null Primitive (for PC deallocation purposes)
E_OFF_PRIM	PA→SEQ(RE)	1	End Of the primitive
FBFACE	PA→SEQ(RE)	1	Front face = 1, back face = 0
TYPE	PA→SEQ(RE)	3	Stippled line and Real time command need to load tex cords from alternate buffer 000 : Normal 001 : Stippled line 011 : Real Time 100 : Line AA 101 : Point AA 110 : Sprite
RTRn	SEQ→PA	1	Stalls the PA in n clocks
RTS	PA→SEQ(RE)	1	PA ready to send data
QuadX	PA→SEQ(RE)	8	Quad X address 2 bits per quad
QuadY	PA→SEQ(RE)	8	Quad Y address 2 bits per quad

~~18.1.3~~20.1.3 VGT to RE : Vertex Bus

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
Vertex indexes	VGT→RE	128	Pointers of indexes or HOS surface information
EOF_vector	VGT→RE	1	End of the vector
Inputs_vert	VGT→RE	1	0: Normal 128 bits per vert 1: double 256 bits per vert
STATE	VGT→SEQ	21	Render State (6*3+3 for constants)

~~18.1.4VGT to SEQ : Vertex Control Bus~~

Formatted: Bullets and Numbering

~~This information needs to be sent over 64 clocks.~~

~~18.1.5~~20.1.4 CP to SEQ : Constant store load

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
Constant Address	CP→SEQ	8	Address of the block of 4 constants
Constant Data	CP→SEQ	512	Data sent over 4 clocks
Remap Address	CP→SEQ	10	Remaping address write address
Remap Data pointer	CP→SEQ	8	Remaping pointer

~~18.1.6~~20.1.5 CP to SEQ : Fetch State store load

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
Constant Address	CP→SEQ	8	Address of the block of 4 state constants

Constant Data	CP→SEQ	512	Data sent over 4 clocks
Remap Address	CP→SEQ	10	Remaping address write address
Remap Data pointer	CP→SEQ	8	Remaping pointer

~~18.1.7~~20.1.6 CP to SEQ : Control State store load

Formatted: Bullets and Numbering

Name	Direction	Bits	Description

{ISSUE: How,Who and what is the size of this bus?}

~~18.1.8~~20.1.7 MH to SEQ: Instruction store Load

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
Instruction address	MH→SEQ	12	Instruction address
Instruction	MH→SEQ	96	Instruction X times
Control Instruction address	MH→SEQ	9	Pointer to the control instruction store
Control Instruction	MH→SEQ	32	Control Instruction X times

~~18.1.9~~20.1.8 SP to RB : Pixel read from RBs

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
Export_data	SP→RB	64	a pair of 32 bits channel values
ExportID	SP→RB	9	0cvvvvhqq: Vertex data vvvv 0-15 from first or second clause (c=0 or 1), XY or ZW components (h=0 or 1), quad 0-3 in the shader (qq= 0-3) 1cbbkttqq: Pixel data for buffer bb (0-3) from first or second clause (0-1) killed or not (k=1 or 0) quad 0-3 in the shader and data is RG (tt=0), BA (tt=1) or Z (tt=2)
ExportMask	SP→RB	2	Specifies whether to write low, high or both 32 bit words. If export mask is 00 data is invalid
ExportLast	SP→RB	1	Last export instruction of the clause

~~18.1.10~~20.1.9 SEQ to RB : Control bus

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
Type	SEQ→RB	1	0: Pixel 1: Vertex
Interleaving	SEQ→RB	1	0: first interleaved clause 1: second interleaved clause
Export_size	SEQ→RB	4	0 thru 16 parameters exported for vertexes (vvvv) OR (bbzs) 1-4 color buffers (bb), two component (s=0) or 4 component colors (s=1) with z (z=1) or without z (z=0)
Valid	SEQ→RB	1	Data valid

Only one exporting clause (7) can be selected at any given time.

~~18.1.11~~20.1.10 RB to SEQ : Output file control

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
Buff_Full	RB→SEQ	1	Set if full
Avail_size	RB→SEQ	6	Size available in output buffers (in 32bits increments)

~~18.1.12~~20.1.11 SP to RB : Position return bus

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
Position return	SP→RB	128	Position data or sprite size (per clock)
Parameter cache pointer	SP→RB	11	Pointer where the data will be in the parameter cache for each vertex

For point sprites and position exports the size and position are interleaved on a 16 x 16 basis. We export 1 position then 1 point sprite sizes. The storage used is of 64x128 bits for position and 64x32 bits for sprite size, it is taken from the output buffer. Additionnally,if needed the edge flags are packed into the bits of the sprite sizes.

~~18.1.13~~20.1.12 *Shader Engine to Fetch Unit Bus (Fast Bus)*

Formatted: Bullets and Numbering

Four quad's worth of addresses is transferred to Fetch Unit every clock. These are sourced from a different pixel within each of the sub-engines repeating every 4 clocks. The register file index to read must precede the data by 2 clocks. The Read address associated with Quad 0 must be sent 1 clock after the Instruction Start signal is sent, so that data is read 3 clocks after the Instruction Start.

Four Quad's worth of Fetch Data may be written to the Register file every clock. These are directed to a different pixel of the sub-engines repeating every 4 clocks. The register file index to write must accompany the data. Data and Index associated with the Quad 0 must be sent 3 clocks after the Instruction Start signal is sent.

Name	Direction	Bits	Description
Tex_Read_Register_Index	SEQ->SP	7	Index into Register files for reading Fetch Address
Tex_RegFile_Read_Data	SP->TEX	2048	16 Fetch Addresses read from the Register file
Tex_Write_Register_Index	SEQ->TEX	7	Index into Register file for write of returned Fetch Data

~~18.1.14~~20.1.13 *Sequencer to Fetch Unit bus (Slow Bus)*

Formatted: Bullets and Numbering

Once every four clock, the fetch unit sends to the sequencer on wich clause it is now working and if the data in the registers is ready or not. This way the sequencer can update the fetch counters for the reservation station fifos. The sequencer also provides the intruction and constants for the fetch to execute and the address in the register file where to write the fetch return data.

Name	Direction	Bits	Description
Tex_Ready	TEX→ SEQ	1	Data ready
Tex_Clause_Num	TEX→ SEQ	3	Clause number
Tex_cst	SEQ→TEX	10	Fetch state address 10 bits sent over 4 clocks
Tex_Inst	SEQ→TEX	12	Fetch instruction address 12 bits sent over 4 clocks
EO_CLAUSE	SEQ→TEX	1	Last instruction of the clause
PHASE	SEQ→TEX	1	Write phase signal

~~19.~~21. Internal interfaces

Formatted: Bullets and Numbering

21.1.1 *RE to SEQ : Vertex Control Bus*

Name	**Direction**	**Bits**	**Description**
STATE	VGT→SEQ	21	Render State (6*3+3 for constants)
Vert counter	VGT→SEQ	6	Which vertices are valid
Inputs_vert	VGT→SEQ	1	0: Normal 128 bits per vert 1: double 256 bits per vert

This information needs to be sent over 64 clocks.

~~20.~~22. Examples of program executions

Formatted: Bullets and Numbering

~~20.1.1~~22.1.1 *Sequencer Control of a Vector of Vertices*

1. PA sends a vector of 64 vertices (actually vertex indices – 32 bits/index for 2048 bit total) to the RE's Vertex FIFO
   - state pointer as well as tag into position cache is sent along with vertices
   - space was allocated in the position cache for transformed position before the vector was sent
   - **also before the vector is sent to the RE, the CP has loaded the global instruction store with the vertex shader program (using the MH?)**
   - The vertex program is assumed to be loaded when we receive the vertex vector.

- the SEQ then accesses the IS base for this shader using the local state pointer (provided to all sequencers by the RBBM when the CP is done loading the program)

2. SEQ arbitrates between the Pixel FIFO and the Vertex FIFO – basically the Vertex FIFO always has priority
   - at this point the vector is removed from the Vertex FIFO
   - the arbitrer is not going to select a vector to be transformed if the parameter cache is full unless the pipe as nothing else to do (ie no pixels are in the pixel fifo).
3. SEQ allocates space in the SP register file for index data plus GPRs used by the program
   - the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer that came down with the vertices
   - SEQ will not send vertex data until space in the register file has been allocated
4. SEQ sends the vector to the SP register file over the RE_SP interface (which has a bandwidth of 2048 bits/cycle)
   - the 64 vertex indices are sent to the 64 register files over 4 cycles
     - RF0 of SU0, SU1, SU2, and SU3 is written the first cycle
     - RF1 of SU0, SU1, SU2, and SU3 is written the second cycle
     - RF2 of SU0, SU1, SU2, and SU3 is written the third cycle
     - RF3 of SU0, SU1, SU2, and SU3 is written the fourth cycle
   - the index is written to the least significant 32 bits **(floating point format?) (what about compound indices)** of the 128-bit location within the register file (w); the remaining data bits are set to zero (x, y, z)
5. SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of fetch state machine 0, or TSM0 FIFO)
   - the control packet contains the state pointer, the tag to the position cache and a register file base pointer.
6. TSM0 accepts the control packet and fetches the instructions for fetch clause 0 from the global instruction store
   - TSM0 was first selected by the TSM arbiter before it could start
7. all instructions of fetch clause 0 are issued by TSM0
8. the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)
   - TSM0 does not wait for requests made to the Fetch Unit to complete; it passes the register file write index for the fetch data to the TU, which will write the data to the RF as it is received
   - once the TU has written all the data to the register files, it increments a counter that is associated with ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go ahead start to execute the ALU clause
9. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store
10. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of fetch state machine 1, or TSM1 FIFO)
11. the control packet continues to travel down the path of reservation stations until all clauses have been executed
    - position can be exported in ALU clause 3 (or 4?); the data (and the tag) is sent over a position bus (which is shared with all four shader pipes) back to the PA's position cache
    - A parameter cache pointer is also sent along with the position data. This tells to the PA where the data is going to be in the parameter cache.
      - there is a position export FIFO in the SP that buffers position data before it gets sent back to the PA
      - the ASM arbiter will prevent a packet from starting an exporting clause if the position export FIFO is full
    - parameter data is exported in clause 7 (as well as position data if it was not exported earlier)
      - parameter data is sent to the Parameter Cache over a dedicated bus
      - the SEQ allocates storage in the Parameter Cache, and the SEQ deallocates that space when there is no longer a need for the parameters (it is told by the PA when using a token).
      - the ASM arbiter will prevent a packet from starting on ASM7 if the parameter cache (or the position buffer if position is being exported) is full
12. after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program

~~20.1.2~~22.1.2 *Sequencer Control of a Vector of Pixels*

Formatted: Bullets and Numbering

1. **As with vertex shader programs, pixel shaders are loaded into the global instruction store by the CP**
   - At this point it is assumed that the pixel program is loaded into the instruction store and thus ready to be read.
2. the RE's Pixel FIFO is loaded with the barycentric coordinates for pixel quads by the detailed walker
   - the state pointer and the LOD correction bits are also placed in the Pixel FIF0
   - the Pixel FIFO is wide enough to source four quad's worth of barycentrics per cycle
3. SEQ arbitrates between Pixel FIFO and Vertex FIFO – when there are no vertices pending OR there is no space left in the register files for vertices, the Pixel FIFO is selected
4. SEQ allocates space in the SP register file for all the GPRs used by the program
   - the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer
   - SEQ will not allow interpolated data to be sent to the shader until space in the register file has been allocated
5. SEQ controls the transfer of interpolated data to the SP register file over the RE_SP interface (which has a bandwidth of 2048 bits/cycle). See interpolated data bus diagrams for details.
6. SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of fetch state machine 0, or TSM0 FIFO)
   - note that there is a separate set of reservation stations/arbiters/state machines for vertices and for pixels
   - the control packet contains the state pointer, the register file base pointer, and the LOD correction bits
   - all other informations (such as quad address for example) travels in a separate FIFO
7. TSM0 accepts the control packet and fetches the instructions for fetch clause 0 from the global instruction store
   - TSM0 was first selected by the TSM arbiter before it could start
8. all instructions of fetch clause 0 are issued by TSM0
9. the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)
   - TSM0 does not wait for fetch requests made to the Fetch Unit to complete; it passes the register file write index for the fetch data to the TU, which will write the data to the RF as it is received
   - once the TU has written all the data for a particular clause to the register files, it increments a counter that is associated with the ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go ahead and pop the FIFO and start to execute the ALU clause
10. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store
11. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of fetch state machine 1, or TSM1 FIFO)
12. the control packet continues to travel down the path of reservation stations until all clauses have been executed
   - pixel data is exported in the last ALU clause (clause 7)
     - it is sent to an output FIFO where it will be picked up by the render backend
     - the ASM arbiter will prevent a packet from starting on ASM7 if the output FIFO is full
13. after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program

~~20.1.3~~22.1.3 *Notes*

Formatted: Bullets and Numbering

14. the state machines and arbitrers will operate ahead of time so that they will be able to immediately start the real threads or stall.
15. the register file base pointer for a vector needs to travel with the vector through the reservation stations, but the instruction store base pointer does not – this is because the RF pointer is different for all threads, but the IS pointer is only different for each state and thus can be accessed via the state pointer

16. Waterfalling, parameter buffer allocation, loops and branches and parameter cache de-allocation still needs to be specked out.

Formatted: Bullets and Numbering

~~21.~~23. Open issues

There is currently an issue with constants. If the constants are not the same for the whole vector of vertices, we don't have the bandwith from the fetch store to feed the ALUs. Two solutions exists for this problem:

1) Let the compiler handle the case and put those instructions in a fetch clause so we can use the bandwith there to operate. This requires a significant amount of temporary storage in the register store.
2) Waterfall down the pipe allowing only at a given time the vertices having the same constants to operate in parrallel. This might in the worst case slow us down by a factor of 16.

Need to do some testing on the size of the register file as well as on the register file allocation method (dynamic VS static).

Saving power?

Size of the fifo containing the information of a vector of pixels/vertices. And size of the fifos before the reservation stations.

Loops and branches.

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 2015~~26~~ October, 200119	GEN-CXXXXX-REVA	1 of 28

Author: Laurent Lefebvre

Issue To:

Copy No:

R400 Sequencer Specification

SEQ

Version ~~0.9~~1.0

Overview: This is an architectural specification for the R400 Sequencer block (SEQ). It provides an overview of the required capabilities and expected uses of the block. It also describes the block interfaces, internal sub-blocks, and provides internal state diagrams.

AUTOMATICALLY UPDATED FIELDS:

Document Location: C:\perforce\r400\arch\doc\gfx\RE\R400_Sequencer.doc

Current Intranet Search Title: R400 Sequencer Specification

APPROVALS	
Name/Dept	Signature/Date

Remarks:

THIS DOCUMENT CONTAINS CONFIDENTIAL INFORMATION THAT COULD BE SUBSTANTIALLY DETRIMENTAL TO THE INTEREST OF ATI TECHNOLOGIES INC. THROUGH UNAUTHORIZED USE OR DISCLOSURE.

Table Of Contents

1. OVERVIEW 4
1.1 Top Level Block Diagram 6
1.2 Data Flow graph 8
1.3 Control Graph 9
2. INTERPOLATED DATA BUS 9
3. INSTRUCTION STORE 12
4. SEQUENCER INSTRUCTIONS 12
5. CONSTANT STORE 12
6. LOOPING AND BRANCHES 12
6.1 The controlling state 12
6.2 The Control Flow Program 13
6.3 Data dependant predicate instructions 16
6.4 Register file indexing 17~~16~~
7. HOS SURFACES 17
8. REGISTER FILE ALLOCATION 17
9. FETCH ARBITRATION 18
10. ALU ARBITRATION 18
11. HANDLING STALLS 19
12. CONTENT OF THE RESERVATION STATION FIFOS 19
13. THE OUTPUT FILE 19
14. IJ FORMAT 19
15. THE PARAMETER CACHE 20
16. VERTEX POSITION EXPORTING 20
17. REAL TIME COMMANDS 20
18. REGISTERS 21~~20~~
18.1 Control 21~~20~~
18.2 Context 21
19. DEBUG REGISTERS 22~~21~~
20. INTERFACES 22~~21~~
20.1 External Interfaces 22~~21~~
20.1.1 PA/SC to RE : IJ bus 22~~21~~
20.1.2 PA/SC to SEQ : IJ Control bus 22~~21~~
20.1.3 VGT to RE : Vertex Bus 23~~22~~
20.1.4 CP to SEQ : Constant store load 23~~22~~
20.1.5 CP to SEQ : Fetch State store load 23~~22~~
20.1.6 CP to SEQ : Control State store load 23~~22~~
20.1.7 MH to SEQ: Instruction store Load 23~~22~~
20.1.8 SP to RB : Pixel read from RBs 23
20.1.9 SEQ to RB : Control bus 24~~23~~
20.1.10 RB to SEQ : Output file control 24~~23~~
20.1.11 SP to RB : Position return bus 25~~23~~
20.1.12 Shader Engine to Fetch Unit Bus (Fast Bus) 25~~24~~
20.1.13 Sequencer to Fetch Unit bus (Slow Bus) 25~~24~~
21. INTERNAL INTERFACES 25~~24~~
21.1.1 RE to SEQ : Vertex Control Bus 25~~24~~
22. EXAMPLES OF PROGRAM EXECUTIONS 26~~24~~
22.1.1 Sequencer Control of a Vector of Vertices 26~~24~~
22.1.2 Sequencer Control of a Vector of Pixels 27~~25~~
22.1.3 Notes 28~~26~~
23. OPEN ISSUES 28~~26~~
~~**1. OVERVIEW 3**~~
~~1.1 Top Level Block Diagram 5~~
~~1.2 Data Flow graph 7~~
~~1.3 Control Graph 8~~
~~**2. INTERPOLATED DATA BUS 8**~~
~~**3. INSTRUCTION STORE 11**~~
~~**4. SEQUENCER INSTRUCTIONS 11**~~
~~**5. CONSTANT STORE 11**~~
~~**6. LOOPING AND BRANCHES 11**~~
~~6.1 The controlling state 11~~
~~6.2 The Control Flow Program 12~~
~~6.3 Data dependant predicate instructions 14~~
~~6.4 Register file indexing 14~~
~~**7. REGISTER FILE ALLOCATION 15**~~
~~**8. FETCH ARBITRATION 16**~~
~~**9. ALU ARBITRATION 16**~~
~~**10. HANDLING STALLS 17**~~
~~**11. CONTENT OF THE RESERVATION STATION FIFOS 17**~~
~~**12. THE OUTPUT FILE 17**~~
~~**13. IJ FORMAT 17**~~
~~**14. THE PARAMETER CACHE 18**~~
~~**15. VERTEX POSITION EXPORTING 18**~~
~~**16. REAL TIME COMMANDS 18**~~
~~**17. REGISTERS 18**~~
~~**18. INTERFACES 19**~~
~~18.1 External Interfaces 19~~
~~18.1.1 PA/SC to RE : IJ bus 19~~
~~18.1.2 PA/SC to SEQ : IJ Control bus 19~~
~~18.1.3 VGT to RE : Vertex Bus 20~~
~~18.1.4 VGT to SEQ : Vertex Control Bus 20~~
~~18.1.5 CP to SEQ : Constant store load 20~~
~~18.1.6 CP to SEQ : Fetch State store load 20~~
~~18.1.7 CP to SEQ : Control State store load 20~~
~~18.1.8 MH to SEQ: Instruction store Load 20~~
~~18.1.9 SP to RB : Pixel read from RBs 20~~
~~18.1.10 SEQ to RB : Control bus 20~~
~~18.1.11 RB to SEQ : Output file control 21~~
~~18.1.12 SP to RB : Position return bus 21~~
~~18.1.13 Shader Engine to Fetch Unit Bus (Fast Bus) 21~~
~~18.1.14 Sequencer to Fetch Unit bus (Slow Bus) 21~~
~~**19. INTERNAL INTERFACES 22**~~
~~**20. EXAMPLES OF PROGRAM EXECUTIONS 22**~~
~~20.1.1 Sequencer Control of a Vector of Vertices 22~~

~~20.1.2 Sequencer Control of a Vector of Pixels 23~~
~~20.1.3 Notes ....24~~
~~21. OPEN ISSUES..24~~

Revision Changes:

Rev 0.1 (Laurent Lefebvre) Date: May 7, 2001	First draft.
Rev 0.2 (Laurent Lefebvre) Date : July 9, 2001	Changed the interfaces to reflect the changes in the SP. Added some details in the arbitration section.
Rev 0.3 (Laurent Lefebvre) Date : August 6, 2001	Reviewed the Sequencer spec after the meeting on August 3, 2001.
Rev 0.4 (Laurent Lefebvre) Date : August 24, 2001	Added the dynamic allocation method for register file and an example (written in part by Vic) of the flow of pixels/vertices in the sequencer.
Rev 0.5 (Laurent Lefebvre) Date : September 7, 2001	Added timing diagrams (Vic)
Rev 0.6 (Laurent Lefebvre) Date : September 24, 2001	Changed the spec to reflect the new R400 architecture. Added interfaces.
Rev 0.7 (Laurent Lefebvre) Date : October 5, 2001	Added constant store management, instruction store management, control flow management and data dependant predication.
Rev 0.8 (Laurent Lefebvre) Date : October 8, 2001	Changed the control flow method to be more flexible. Also updated the external interfaces.
Rev 0.9 (Laurent Lefebvre) Date : October 17, 2001	Incorporated changes made in the 10/18/01 control flow meeting. Added a NOP instruction, removed the conditional_execute_or_jump. Added debug registers.
Rev 1.0 (Laurent Lefebvre) Date : October 19, 2001	Refined interfaces to RB. Added state registers.

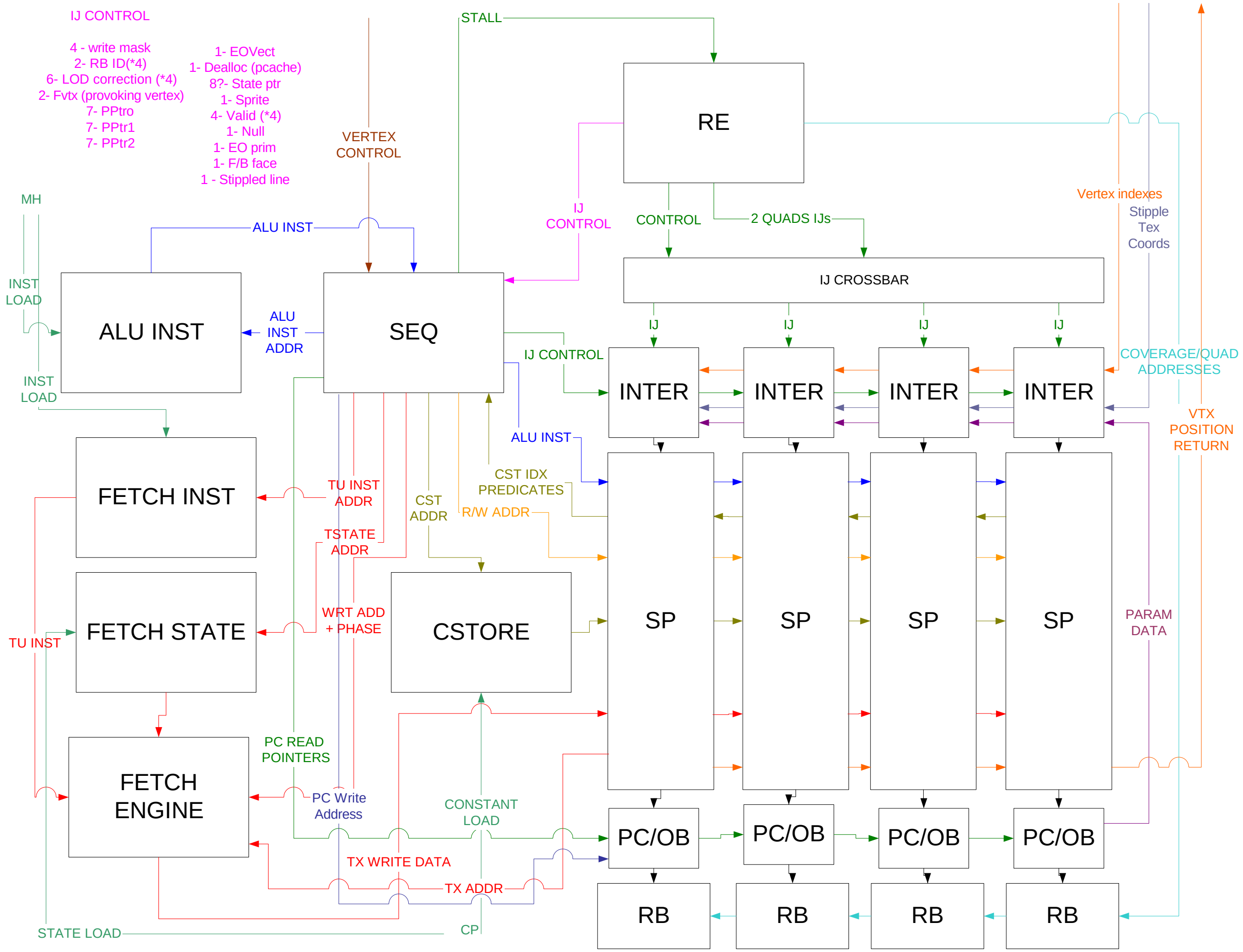
1. Overview

The sequencer first arbitrates between vectors of 64 vertices that arrive directly from primitive assembly and vectors of 16 quads (64 pixels) that are generated in the raster engine.

The vertex or pixel program specifies how many GPR's it needs to execute. The sequencer will not start the next vector until the needed space is available.

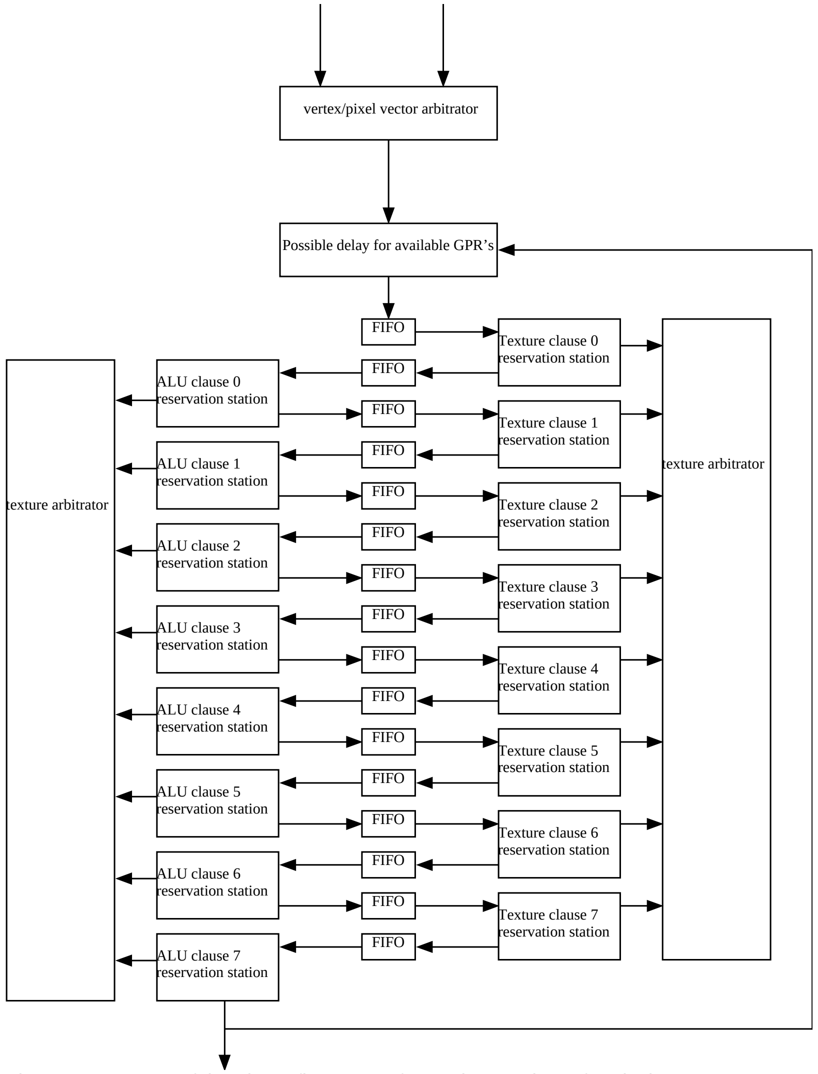
The sequencer is based on the R300 design. It chooses two ALU clauses and a fetch clause to execute, and executes all of the instructions in a clause before looking for a new clause of the same type. Two ALU clauses are executed interleaved to hide the ALU latency. Each vector will have eight fetch and eight ALU clauses, but clauses do not need to contain instructions. A vector of pixels or vertices ping-pongs along the sequencer FIFO, bouncing from fetch reservation station to alu reservation station. A FIFO exists between each reservation stage, holding up vectors until the vector currently occupying a reservation station has left. A vector at a reservation station can be chosen to execute. The sequencer looks at all eight alu reservation stations to choose an alu clause to execute and all eight fetch stations to choose a fetch clause to execute. The arbitrator will give priority to clauses/reservation stations closer to the bottom of the pipeline. It will not execute an alu clause until the fetch fetches initiated by the previous fetch clause have completed. There are two separate sets of reservation stations, one for pixel vectors and one for vertices vectors. This way a pixel can pass a vertex and a vertex can pass a pixel.

To support the shader pipe the raster engine also contains the shader instruction cache and constant store. There are only one constant store for the whole chip and one instruction store. These will be shared among the four shader pipes. The four shader pipes also execute the same instuction thus there is only one sequencer for the whole chip.



IJ CONTROL
4 - write mask
2- RB ID(*4)
6- LOD correction (*4)
2- Fvtx (provoking vertex)
7- PPtro
7- PPtr1
7- PPtr2
1- EOVect
1- Dealloc (pcache)
8?- State ptr
1- Sprite
4- Valid (*4)
1- Null
1- EO prim
1- F/B face
1 - Stippled line
VERTEX CONTROL
STALL
RE
IJ CONTROL
CONTROL
2 QUADS IJs
Vertex indexes
Stipple Tex Coords
MH
ALU INST
INST LOAD
ALU INST
ALU INST ADDR
SEQ
IJ CROSSBAR
IJ
IJ
IJ
IJ
IJ CONTROL
INTER
INTER
INTER
INTER
COVERAGE/QUAD ADDRESSES
INST LOAD
ALU INST
VTX POSITION RETURN
FETCH INST
TU INST ADDR
CST ADDR
CST IDX PREDICATES
R/W ADDR
TSTATE ADDR
SP
SP
SP
SP
PARAM DATA
TU INST
FETCH STATE
WRT ADD + PHASE
CSTORE
PC READ POINTERS
FETCH ENGINE
PC Write Address
CONSTANT LOAD
PC/OB
PC/OB
PC/OB
PC/OB
TX WRITE DATA
TX ADDR
RB
RB
RB
RB
STATE LOAD
CP

1.1 Top Level Block Diagram


vertex/pixel vector arbitrator
Possible delay for available GPR's
FIFO
Texture clause 0 reservation station
Texture clause 1 reservation station
Texture clause 2 reservation station
Texture clause 3 reservation station
Texture clause 4 reservation station
Texture clause 5 reservation station
Texture clause 6 reservation station
Texture clause 7 reservation station
texture arbitrator
ALU clause 0 reservation station
ALU clause 1 reservation station
ALU clause 2 reservation station
ALU clause 3 reservation station
ALU clause 4 reservation station
ALU clause 5 reservation station
ALU clause 6 reservation station
ALU clause 7 reservation station
texture arbitrator

There are two sets of the above figure, one for vertices and one for pixels.

~~The rasterizer always checks the vertices FIFO first and if allowed by the sequencer sends the data to the shader. If the vertex FIFO is empty then, the rasterizer takes the first entry of the pixel FIFO (a vector of 64 pixels) and sends it to the interpolators. Then the sequencer takes control of the packet~~Depending on the arbitration state, the sequencer will either choose a vertex or a pixel packet. The control packet consists of 21 bits of state, 6-7 bits for the base address of the Shader program and some information on the coverage to determine fetch LOD plus other various small state bits.

On receipt of a packet, the input state machine (not pictured but just before the first FIFO) allocated enough space in the registers to store the interpolated values and temporaries. Following this, the input state machine stacks the packet in the first FIFO.

On receipt of a command, the level 0 fetch machine issues a texure request and corresponding register address for the fetch address (ta). A small command (tcmd) is passed to the fetch system identifying the current level number (0) as well as the register write address for the fetch return data. One fetch request is sent every 4 clocks causing the texturing of sixteen 2x2s worth of data (or 64 vertices). Once all the requests are sent the packet is put in FIFO 1.

Upon recept of the return data, the fetch unit writes the data to the register file using the write address that was provided by the level 0 fetch machine and sends the clause number (0) to the level 0 fetch state machine to signify that the write is done and thus the data is ready. Then, the level 0 fetch machine increments the counter of FIFO 1 to signify to the ALU 1 that the data is ready to be processed.

On receipt of a command, the level 0 ALU machine first decrements the input FIFO counter and then issues a complete set of level 0 shader instructions. For each instruction, the state machine generates 3 source addresses, one destination address (3 cycles later) and an instruction. Once the last instruction as been issued, the packet is put into FIFO 2.

There will always be two active ALU clauses at any given time (and two arbitrers). One arbitrer will arbitrate over the odd instructions (4 clocks cycles) and the other one will arbitrate over the even instructions (4 clocks cycles). The only constraints between the two arbitrers is that they are not allowed to pick the same clause number as the~~y~~ other one is currently working on if the packet ~~os~~ is not of the same type (render state).

If the packet is a vertex packet, upon reaching ALU clause 3, it can export the position if the position is ready. So the arbitrer must prevent ALU clause 3 to be selected if the positional buffer is full (or can't be accessed). Along with the positional data, the location where the vertex data is to be put is also sent (parameter data pointers).

{ISSUE: How do we handle parameter cache pointers (computed, semi-computed or not computed)?}

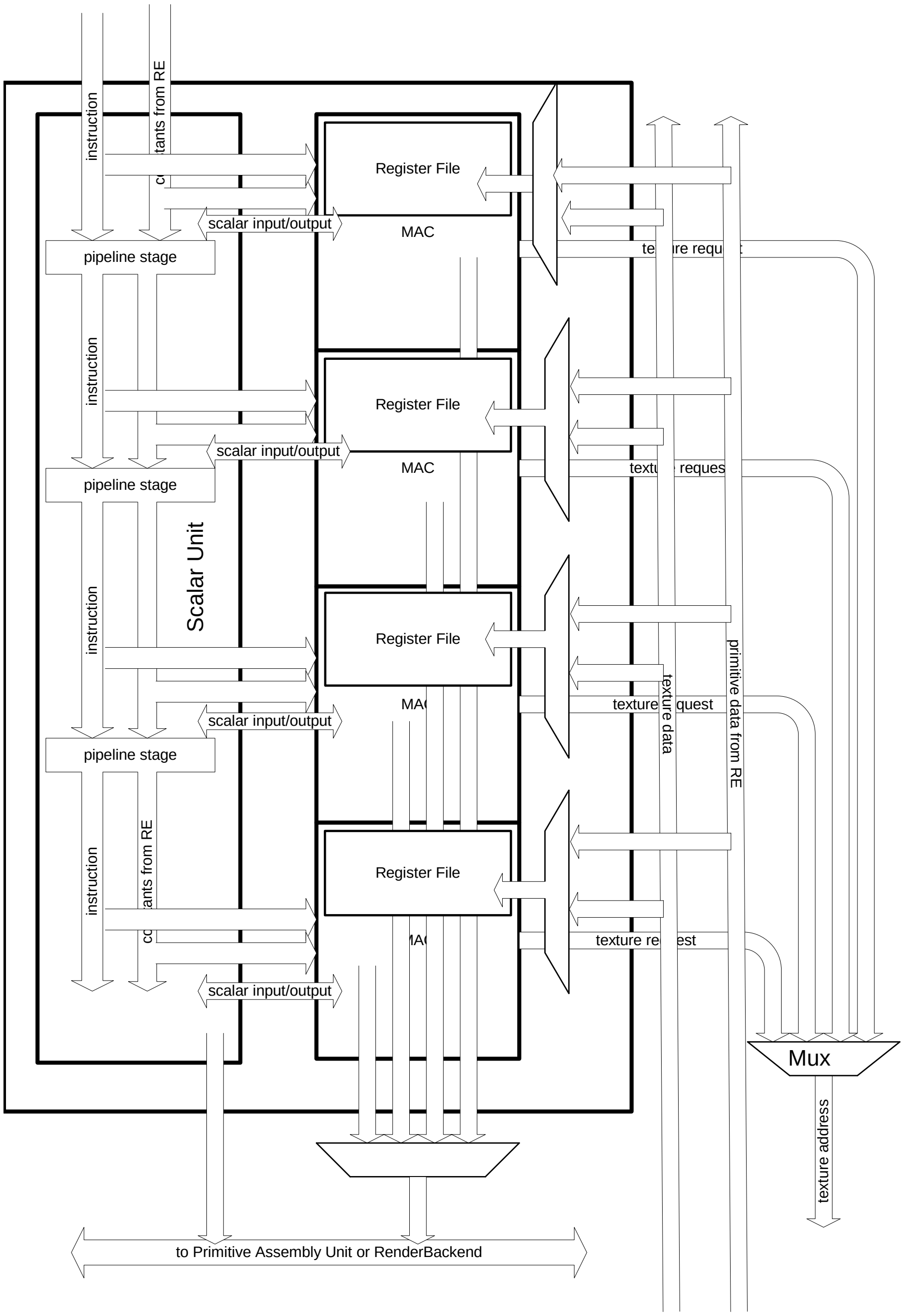
A special case is for HOS surfaces wich can export 12 parameters per last 6 clauses to the output buffer. If the output buffer is full or doesn't have enough space the sequencer will prevent such a vertex group to enter an exporting clause.

Regular pixel and vertex shaders can export 12 parameters to memory from the last clause only (7).

All other level process in the same way until the packet finally reaches the last ALU machine (~~8~~7). On completion of the level ~~8~~ 7 ALU clause, a valid bit is sent to the Render Backend which picks up the color data. This requires that the last instruction writes to the output register – a condition that is almost always true. If the packet was a vertex packet, instead of sending the valid bit to the RB, it is sent to the PA so it can know that the data present in the parameter store is valid.

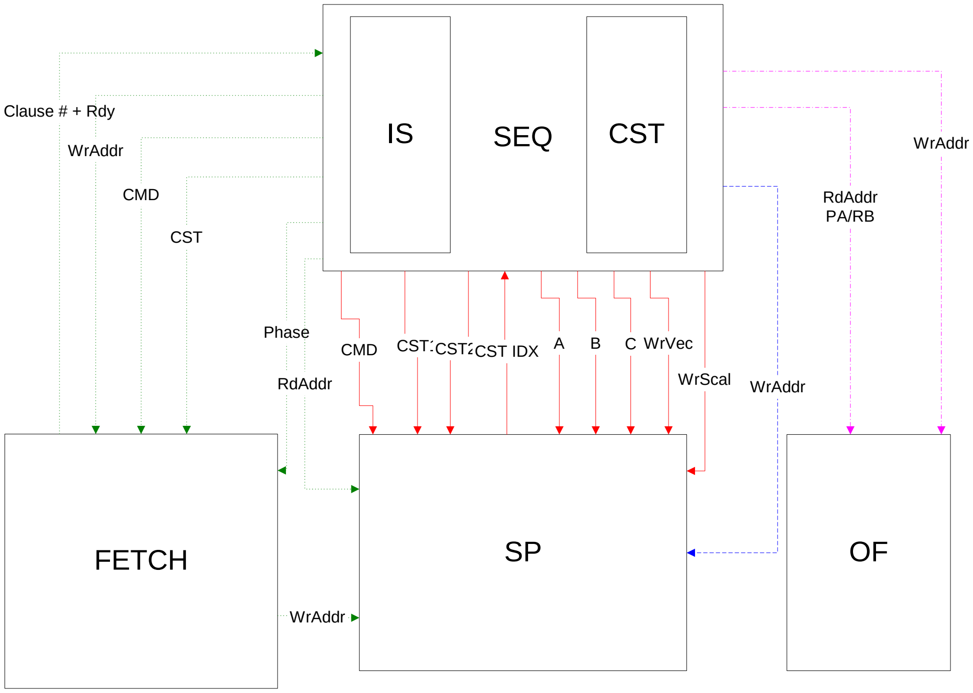
Only two ALU state machine may have access to the register file address bus or the instruction decode bus at one time. Similarly, only one fetch state machine may have access to the register file address bus at one time. Arbitration is performed by three arbitrer blocks (two for the ALU state machines and one for the fetch state machines). The arbitrers always favor the higher number state machines, preventing a bunch of half finished jobs from clogging up the register files.

1.2 Data Flow graph



The gray area represents blocks that are replicated 4 times per shader pipe (16 times on the overall chip).

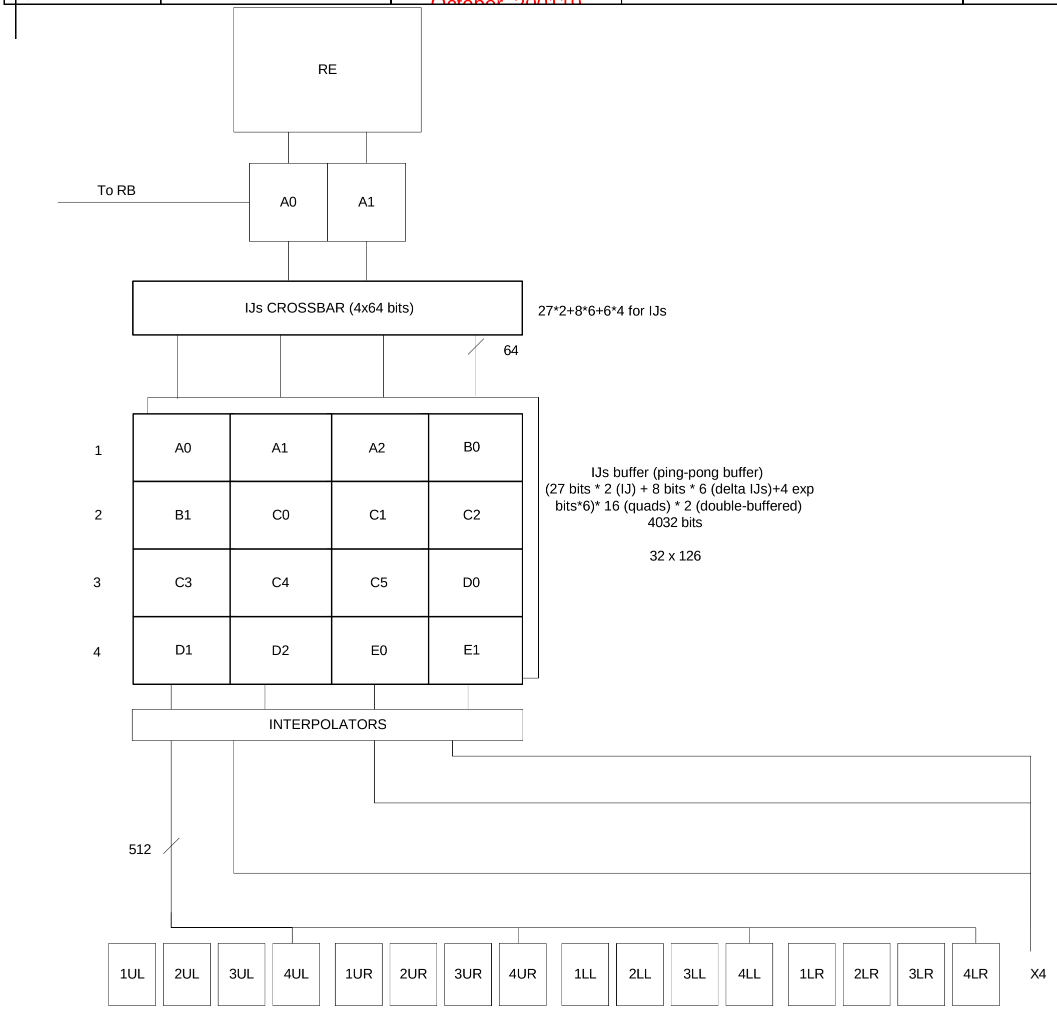
1.3 Control Graph



In green is represented the Fetch control interface, in red the ALU control interface, in blue the Interpolated/Vector control interface and in purple is the output file control interface.

2. Interpolated data bus

The interpolators contain an IJ buffer to pack the information as much as possible before writing it to the register file.



RE
To RB
A0
A1
IJs CROSSBAR (4x64 bits)
27*2+8*6+6*4 for IJs
64
1
A0
A1
A2
B0
2
B1
C0
C1
C2
3
C3
C4
C5
D0
4
D1
D2
E0
E1
IJs buffer (ping-pong buffer)
(27 bits * 2 (IJ) + 8 bits * 6 (delta IJs)+4 exp bits*6)* 16 (quads) * 2 (double-buffered)
4032 bits
32 x 126
INTERPOLATORS
512
1UL
2UL
3UL
4UL
1UR
2UR
3UR
4UR
1LL
2LL
3LL
4LL
1LR
2LR
3LR
4LR
X4

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19
SP0	A0	B1	C3	D1					A0	B1	C3	D1					V 0-3	V 16-19	V 32-35	V 48-51
SP1	A1		C4	D2		C0			A1		C4	D2		C0			V 4-7	V 20-23	V 36-39	V 52-55
SP2	A2		C5			C1		E0	A2		C5			C1		E0	V 8-11	V 24-27	V 40-43	V 56-59
SP3					B0	C2	D0	E1					B0	C2	D0	E1	V 12-15	V 28-31	V 44-47	V 60-63

P0

P1

Above is an example of a tile we might receive. The IJ information is packed in the IJ buffer 2 quads at a time. The sequencer allows at any given time as many as four quads to interpolate a parameter. They all have to come from the same primitive. Then the sequencer controls the write mask to the register to write the valid data in.

3. Instruction Store

There is going to be only one instruction store for the whole chip. It will contain 4096 instructions of 96 bits each. There is also going to be a control instruction store of size 256(512?)x32.

{ISSUE : The instruction store is loaded by the sequencer using the memory hub ?}.

The read bandwith from this store is 96*2 bits/ 4 clocks (48 bits/clock). It is likely to be a 1 port memory; we use 1 clock to load the ALU instruction, 1 clocks to load the Fetch instruction, 1 clock to load 2 control flow instructions and 1 clock to write instructions.

4. Sequencer Instructions

All control flow instructions and move instructions are handled by the sequencer only. The ALUs will perform NOPs during this time (MOV PV,PV, PS,PS).

5. Constant Store

The constant store is managed by the CP. The sequencer is aware of where the constants are using a remaping table also managed by the CP. A likely size for the constant store is 512x128 bits. The constant store is also planned to be shared. The read BW from the constant store is 128 bits/clock and the write bandwith is 32/4 bits/clock.

In order to do constant store indexing, the sequencer must be loaded first with the indexes (that come from the GPRs). There are 144 wires from the exit of the SP to the sequencer (9 bits pointers x 16 vertexes/clock). Since the data must pass thru the Shader pipe for the float to fixed convertion, there is a latency of 4 clocks (1 instruction) between the time the sequencer is loaded and the time one can index into the constant store. The assembly will look like this

```
MOVA  R1.X,R2.X       // Loads the sequencer with the content of R2.X, also copies the content of R2.X into R1.X
NOP                   // latency of the float to fixed conversion
ADD   R3,R4,C0[R2.X]// Uses the state from the sequencer to add R4 to C0[R2.X] into R3
```

Note that we don't really care about what is in the brackets because we use the state from the MOVA instruction. R2.X is just written again for the sake of simplicity.

The storage needed in the sequencer in order to support this feature is 2*64*9 bits = 1152 bits.

6. Looping and Branches

Loops and branches are planned to be supported and will have to be dealt with at the sequencer level. We plan on supporting constant loops and branches using a control program.

6.1 The controlling state.

As per Dx9 the following state is available for control flow:

Boolean[15:0]
loop_count[7:0][7:0]
        In addition:
loop_start [7:0] [7:0]
loop_step [7:0] [7:0]
        Exist to give more control to the controlling program.

We will extend that in the R400 to:

Boolean[255:0]
Loop_count[7:0][15:0]
Loop_Start[7:0] [15:0] times 2 (one for constant,registert)
Loop_Step[7:0] [15:0] times 2 (one for constant,register)
Loop_End[7:0] [15:0]

{ISSUE: How is the controlling state loaded and how many contexts do we have?}

We have a stack of 4 elements for calling subroutines and 4 loop counters to allow for nested loops.

We also keep 8 predicate vectors and 8 AND/OR sets of 3 bits. These bits can be 0: all 0s, 1: all ones and 11: mixed.

6.2 The Control Flow Program

The R300 uses a match method for control flow: The shader is executed, and at every instruction its address is compared with addresses (or address?) in a control table. The “event” in the control table can redirect operations in the program.

The Method chosen for the R400 is a “control program”. The control program has ten basic instructions:

Execute
Conditional_execute
Conditional_Execute_Predicates
Conditional_execute_or_Jump
Conditional_jump
Call
Return
Loop_start
Loop_end
End_of_clause

Execute, causes the specified number of instructions in instruction store to be executed.
Conditional_execute checks a condition first, and if true, causes the specified number of instructions in instruction store to be executed.
Loop_start resets the corresponding loop counter to the start value on the first pass after it checks for the end condition and if met jumps over to a specified address.
Loop_end increments (decrements?) the loop counter and jumps back the specified number of instructions.
Call jumps to an address and pushes the IP counter on the stack. On the return instruction, the IP is poped from the stack.
Conditional_execute_or_Jump executes a block of instructions or jumps to an address is the condition is not met.
Conditional_execute_Predicates executes a block of instructions if all bits in the predicate vectors meet the condition.
End_of_clause marks the end of a clause.
Conditional_jumps jumps to an address if the condition is met.
NOP is a regular NOP

NOTE THAT ALL JUMPS MUST JUMP TO EVEN CFP ADDRESSES. Thus the compiler must insert NOPs where needed to align the jumps on even CFP addresses.

Also if the jump is logically bigger than ~~4096~~ pshader_cntl_size (or vshader_cntl_size) we break the program (clause) and set the debug registers. If an execute or conditional_execute is lower than cntl_size or bigger than size we also break the program (clause) and set the debug registers.

We have to fit instructions into 48 bits in order to be able to put two control flow instruction per line in the instruction store.

Execute				
47	46… 42	41 … 24	23 … 12	11 … 0
Addressing	00001	RESERVED	Instruction _count	Exec Address

Execute up to 4k instructions at the specified address in the instruction memory.

NOP		
47	46 … 42	41 … 0
Addressing	00010	RESERVED

If the specified boolean (8 bits can address 256 booleans) meets the specified condition then execute the specified instructions (up to 512 instructions) or if the condition is not met jump to the jump address in the control flow program. This MUST be a forward jump.

Conditionnal_Execute						
47	46 … 42	41 … 34	33	32 … 24	23 … 12	11 … 0
Addressing	00011	Boolean address	Condition	RESERVED	Instruction_count	Exec Address

If the specified boolean (8 bits can address 256 booleans) meets the specified condition then execute the specified instructions (up to 4k instructions)

Conditionnal_Execute_Predicates						
47	46 … 42	41 … 38	37	36 … 24	23 … 12	11 … 0
Addressing	00100	Predicate vector	Condition	RESERVED	Instruction_count	Exec Address

Check the AND/OR of all current predicate bits. If AND/OR matches the condition execute the specified number of instructions.

Loop_Start				
47	46 … 42	41 … 16	15 … 4	3 … 0
Addressing	00101	RESERVED	Jump address	Loop ID

Loop Start. Compares the loop count with the end value. If loop condition not met jump to the address. Forward jump only. Also computes the index value.

Loop_End				
47	46 … 42	41 … 16	15 … 4	3 … 0
Addressing	00111	RESERVED	Start address	Loop ID

Loop end. Increments the counter by one and jumps BACK only to the start of the loop.

The way this is described does not prevent nested loops, and the inclusion of the loop id make this easy to do.

Call			
47	46 … 42	41…12	11 … 0
Addressing	01000	RESERVED	Address

Jumps to the specified address and pushes the IP counter on the stack.

Return		
47	46 … 42	41 … 0
Addressing	01001	RESERVED

Pops the topmost address from the stack and jumps to that address. If nothing is on the stack, the program will just continue to the next instruction.

Conditional_Jump						
47	46 … 42	41 … 34	33	32 … 13~~2~~	12	11 … 0
Addressing	01010	Boolean address	Condition	RESERVED	FW only	Address

If condition met, jumps to the address. FORWARD jump only allowed if bit 12 set. Bit 12 is only an optimization for the compiler and should NOT be exposed to the API.

End_of_Clause		
47	46 … 42	41 … 0

Addressing	01011	RESERVED

Marks the end of a clause.

To prevent infinite loops, we will keep 9 bits loop counters instead of 8 (we are only able to loop 256 times). If the counter goes higher than 255 then the loop_end or the loop_start is going to break the loop and set de debug registers. The sequencer will keep two loop indexes values:

IC index for constant indexing (9 bits)
IR index for register file indexing (7 bits)

This will be updated everytime we loop and can only be used to index the constant store and the register file. The way to compute this value is:

Index = Loop_counter*Loop_iterator + Loop_init.

The IC for constant is going to return 0 if it is out of the constant range. The IR index is going to break the program if the index exeeds the number of requested registers.

The basic model is as follows:

The render state defined the clause boundaries:

Vertex_shader_fetch[7:0][7:0]	// eight 8 bit pointers to the location where each clauses control program is located
Vertex_shader_alu[7:0][7:0]	// eight 8 bit pointers to the location where each clauses control program is located
Pixel_shader_fetch[7:0][7:0]	// eight 8 bit pointers to the location where each clauses control program is located
Pixel_shader_alu[7:0][7:0]	// eight 8 bit pointers to the location where each clauses control program is located

The control program for a given clause is executed to completion before moving to another clause, (with the exception of the pick two nature of the alu execution). The control program is the only program aware of the clause boundaries.

6.3 Data dependant predicate instructions

Data dependant conditionals will be supported in the R400. The only way we plan to support those is by supporting three vector/scalar predicate operations of the form:

PRED_SETE_# - similar to SETE except that the result is 'exported' to the sequencer.
PRED_SETGT_# - similar to SETGT except that the result is 'exported' to the sequencer
PRED_SETGTE_# - similar to SETGTE except that the result is 'exported' to the sequencer

For the scalar operations only we will also support the two following instructions:

PRED_SETE0_# – SETE0
PRED_SETE1_# – SETE1

The export is a single bit - 1 or 0 that is sent using the same data path as the MOVA instruction. The sequencer will maintain 4 sets of 64 bit predicate vectors (in fact 8 sets because we interleave two programs but only 4 will be exposed) and use it to control the write masking. This predicate is not maintained across clause boundaries. The # sign is used to specify wich predicate set you want to use 0 thru 3.

Then we have two conditional execute bits. The first bit is a conditional execute "on" bit and the second bit tells us if we execute on 1 or 0. For exemple, the ~~instruction :~~instruction:

P0_ADD_# R0,R1,R2

Is only going to write the result of the ADD into those GPRs whose predicate bit is 0. Alternatively, P1_ADD_# would only write the results to the GPRs whose predicate bit is set. The use of the P0 or P1 without precharging the sequencer with a PRED instruction is undefined.

{Issue: do we have to have a NOP between PRED and the first instruction that uses a predicate?}

6.4 Register file indexing

Because we can have loops in fetch clause, we need to be able to index into the register file in order to retrieve the data created in a fetch clause loop and use it into an ALU clause. The instruction will include the base address for register indexing and the instruction will contain these controls :

Bit7	Bit 6	
0	0	'absolute register'
0	1	'relative register'
1	0	'previous vector'
1	1	'previous scalar'

In the case of an absolute register we just take the address as is. In the case of a relative register read we take the base address and we add to it the loop_index and this becomes our new address that we give to the shader pipe.

Formatted: Bullets and Numbering

7. Pixel Kill Mask

A vector of 64 bits is kept per group of pixels/vertices. Its purpose is to optimize the texture fetch requests and allow the shader pipe to kill pixels using the following instructions:

MASK_SETE
MASK_SETGT
MASK_SETGTE

However, if the driver sets the kill_vector_on register to 0 (don't use) then the 64 bit kill mask becomes the 5th predicate vector and is kept across clause boundaries (thus allowing predicated instructions to be used in texture clauses). In this mode, the sequencer is going to send all 1s to the RBs for coverage mask information.

Formatted

Formatted: Bullets and Numbering

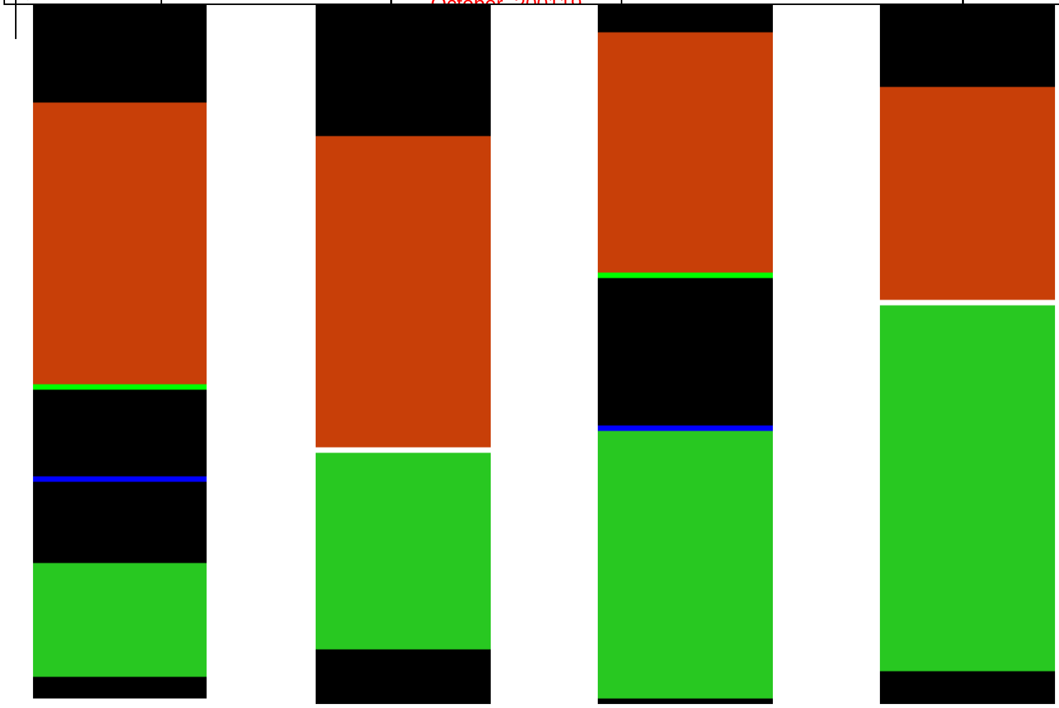
~~7.~~8. HOS surfaces

HOS surfaces are able to export from ~~any~~ the 6 last clauses but to memory ONLY. If they want to export to the parameter cache they have to do it in the last clause (7). They can also export position in clause 3. The buffer they want to export into must be specified in the "exports" field of the state registers.

Formatted: Bullets and Numbering

~~8.~~9. Register file allocation

The register file allocation for vertices and pixels can either be static or dynamic. In both cases, the register file in managed using two round robins (one for pixels and one for vertices). In the dynamic case the boundary between pixels and vertices is allowed to move, in the static case it is fixed to VERTEX_REG_SIZE for vertices and 256-VERTEX_REG_SIZE for pixels.



Above is an example of how the algorithm works. Vertices come in from top to bottom; pixels come in from bottom to top. Vertices are in orange and pixels in green. The blue line is the tail of the vertices and the green line is the tail of the pixels. Thus anything between the two lines is shared. When pixels meets vertices the line turns white and the boundary is static until both vertices and pixels share the same “unallocated bubble”. Then the boundary is allowed to move again.

Formatted: Bullets and Numbering

~~9.~~10. Fetch Arbitration

The fetch arbitration logic chooses one of the 8 potentially pending fetch clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute. Once chosen, the clause state machine will send one 2x2 fetch per clock (or 4 fetches in one clock every 4 clocks) until all the fetch instructions of the clause are sent. This means that there cannot be any dependencies between two fetches of the same clause.

The arbitrator will not wait for the fetches to return prior to selecting another clause for execution. The fetch pipe will be able to handle up to X(?) in flight fetches and thus there can be a fair number of active clauses waiting for their fetch return data.

Formatted: Bullets and Numbering

~~10.~~11. ALU Arbitration

ALU arbitration proceeds in almost the same way than fetch arbitration. The ALU arbitration logic chooses one of the 8 potentially pending ALU clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute. There are two ALU arbitrers, one for the even clocks and one for the odd clocks. For exemple, here is the sequencing of two interleaved ALU clauses (E and O stands for Even and Odd sets of 4 clocks):

Einst0 Oinst0 Einst1 Oinst1 Einst2 Oinst2 Einst0 Oinst3 Einst1 Oinst4 Einst2 Oinst0…
Proceeding this way hides the latency of 8 clocks of the ALUs.

~~11.~~12. Handling Stalls

Formatted: Bullets and Numbering

When the output file is full, the sequencer prevents the ALU arbitration logic to select the last clause (this way nothing can exit the shader pipe until there is place in the output file. If the packet is a vertex packet and the position buffer is full (POS_FULL) then the sequencer also prevents a thread to enter the exporting clause (4?). The sequencer will set the OUT_FILE_FULL signal n clocks before the output file is actually full and thus the ALU arbitrer will be able read this signal and act accordingly by not preventing exporting clauses to proceed.

~~12.~~13. Content of the reservation station FIFOs

Formatted: Bullets and Numbering

21 bits of Render State 7 bits for the base address of the GPRs, some bits for LOD correction and coverage mask information in order to fetch fetch for only valid pixels, quad address and 1 bit to specify if the vector is of pixels or vertices. Since pixels and vertices are kept in order in the shader pipe, we only need two fifos (one for vertices and one for pixels) deep enough to cover the shader pipe latency. This size will be determined later when we will know the size of the small fifos between the reservation stations.

~~For texture clauses, 3 bits * 4 are going to be kept. These are the AND/OR of the predicate vectors. 0 for all 0s, 1 for all ones and MIXED.~~

~~13.~~14. The Output File

Formatted: Bullets and Numbering

The output file is where pixels are put before they go to the RBs. The write BW to this store is 256 bits/clock. Just before this output file are staging registers with write BW 512 bits/clock and read BW 256 bits/clock. For this reason only ONE concurrent program can be of clause 8 (exporting clause) the other program MUST not. The staging registers are 4x128 (and there are 16 of those on the whole chip).

~~14.~~15. IJ Format

Formatted: Bullets and Numbering

The IJ information sent by the PA is of this format on a per quad basis:

We have a vector of IJ's (one IJ per pixel at the centroid of the fragment or at the center of the pixel depending on the mode bit). The interpolation is done at a different precision across the 2x2. The upper left pixel's parameters are always interpolated at full 19x24 mantissa precision. Then the result of the interpolation along with the difference in IJ in reduced precision is used to interpolate the parameter for the other three pixels of the 2x2. Here is how we do it:

Assuming P0 is the interpolated parameter at Pixel 0 having the barycentric coordinates I(0), J(0) and so on for P1,P2 and P3. Also assuming that A is the parameter value at V0 (interpolated with I), B is the parameter value at V1 (interpolated with J) and C is the parameter value at V2 (interpolated with (1-I-J).

$$\Delta 01I = I(1) - I(0)$$
$$\Delta 01J = J(1) - J(0)$$
$$\Delta 02I = I(2) - I(0)$$
$$\Delta 02J = J(2) - J(0)$$
$$\Delta 03I = I(3) - I(0)$$
$$\Delta 03J = J(3) - J(0)$$

P0	P1
P2	P3

$$P0 = C + I(0) * (A - C) + J(0) * (B - C)$$
$$P1 = P0 + \Delta 01I * (A - C) + \Delta 01J * (B - C)$$
$$P2 = P0 + \Delta 02I * (A - C) + \Delta 02J * (B - C)$$
$$P3 = P0 + \Delta 03I * (A - C) + \Delta 03J * (B - C)$$

P0 is computed at 19x24 mantissa precision and P1 to P3 are computed at 8X24 mantissa precision. So far no visual degradation of the image was seen using this scheme.

Multiples (Full Precision): 2
Multiples (Reduced precision): 6
Subtracts 19x24 (Parameters): 2
Adds: 8

FORMAT OF P0's IJ : Mantissa 19 Exp 4 for I + Sign
Mantissa 19 Exp 4 for J + Sign

FORMAT of Deltas (x3):Mantissa 8 Exp 4 for I + Sign
Mantissa 8 Exp 4 for J + Sign

Total number of bits : 19*2 + 8*6 + 4*8 + 4*2 = 126

Formatted: Bullets and Numbering

~~15.~~16. The parameter cache

The parameter cache is where the vertex shaders export their data. It consists of 16 128x128 memories (1R/1W). The reuse engine will make it so that all vertexes of a given primitive will hit different memories.

Formatted: Bullets and Numbering

~~16.~~17. Vertex position exporting

On clause 4 (or 5) the vertex shader can export to the PA both the vertex position and the point sprite. It can also do so at clause 8 if not done at clause 4. Along with the position is exported a pointer to the parameter cache where the data will be once the vertex shader exports. The storage needed to perform the position export is at least 64x128 memories for the position and 64x32 memories for the sprite size. It is going to be taken in the pixel output fifo.

Formatted: Bullets and Numbering

18. Exporting Arbitration

Here are the rules for co-issuing exporting ALU clauses.

Formatted: Bullets and Numbering

1) Position exports and position exports cannot be co-issued.
2) Position exports and memory exports cannot be co-issued.
3) Position exports and Z/Color exports cannot be co-issued.
4) Memory exports and Z/Color exports cannot be co-issued.
5) Memory exports and memory exports cannot be co-issued.
6) Z/color exports and Z/color exports cannot be co-issued.
7) Parameter exports and Z/Color exports CAN be co-issued.
8) Parameter exports and parameter exports CAN be co-issued.
~~1)~~9)Parameter exports and memory exports CAN be co-issued.

~~17.~~19. Real time commands

We are unable to use the parameter memory since there is no way for a command stream to write into it. Instead we need to add three 16x128 memories (one for each of three vertices x 16 interpolants). These will be mapped onto the register bus and written by type 0 packets, and output to the the parameter busses (the sequencer and/or PA need to be able to address the realtime parameter memory as well as the regular parameter store. For higher performance we should be able able to view them as two banks of 16 and do double buffering allowing one to be loaded, while the other is rasterized with. Most overlay shaders will need 2 or 4 scalar coordinates, one option might be to restrict the memory to 16x64 or 32x64 allowing only two interpolated scalars per cycle, the only problem I see with this is, if we view support for 16 vector-4 interpolants important (true only if we map microsoft's high priority stream to the realtime stream), then the PA/sequencer need to support a realtime-specific mode where we need to address 32 vectors of parameters instead of 16.

~~18.~~20. Registers

Formatted: Bullets and Numbering

20.1 Control

DYNAMIC_REG	Dynamic allocation (pixel/vertex) of the register file on or off.
VERTEX_REG_SIZE	What portion of the register file is reserved for vertices (static allocation only)
PIXEL_MIN_SIZE	Minimal size of the register file's pixel portion (dynamic only)
VERTEX_MIN_SIZE	Minimal size of the register file's vertex portion (dynamic only)
ARBITRATION_policy	policy of the arbitration between vetexes and pixels
CST_SIZE_P	Size of the constant store for pixels
CST_SIZE_V	Size of the constant store for vertexes
INST_STOR_ALLOC	interleaved, separate, interleaved+shared,separate+shared
VWRAP	wrap point for the vertex shader instruction store
PWRAP	wrap point for the pixel shader instruction store
NO_INTERLEAVE	debug state register. Only allows one program at a time into the GPRs

~~18.1~~20.2 Context

Formatted: Bullets and Numbering

Vshader_fetch[~~11~~7:0][7:0]	eight ~~12~~ 8 bit pointers to the location where each clauses control program is located
Vshader_alu[~~11~~7:0][7:0]	eight ~~12~~ 8 bit pointers to the location where each clauses control program is located
Pshader_fetch[~~11~~7:0][7:0]	eight ~~12~~ 8 bit pointers to the location where each clauses control program is located
Pshader_alu[~~11~~7:0][7:0]	eight ~~12~~ 8 bit pointers to the location where each clauses control program is located
PSHADER	base pointer for the pixel shader
VSHADER	base pointer for the vertex shader
Vshader_cntl_size	size of the vertex shader (# of instructions in control program/2)
Pshader_cntl_size	size of the pixel shader (# of instructions in control program/2)
Pshader_size	size of the pixel shader (cntl+instructions)
Vshader_size	size of the vertex shader (cntl+instructions)
~~VWRAP~~	~~wrap point for the vertex shader instruction store~~
~~PWRAP~~	~~wrap point for the pixel shader instruction store~~
REG_ALLOC_PIX	number of registers to allocate for pixel shader programs
REG_ALLOC_VERT	number of registers to allocate for vertex shader programs
~~PARAM_MASK[0…16]~~	~~parameter mask to specify how parameters maps in the pixel shader~~
FLAT_GOUR[0…1~~6~~5]	wich parameters are to be gouraud shaded
GEN_TEX~~[0….16]~~	~~for wich parameters do we need to generate tex coords.~~Do we generate texture coordinates for 1st parameter or not
CYL_WRAP[0…~~64~~63]	for wich parameters (and channels (xyzw)) do we do the cyl wrapping.
P_export_mode	0xxxx : Normal mode
	1xxxx : Multipass mode
	If normal, bbbz where bbb is how many colors (0-4) and z is export z or not
	If multipass 1-12 exports for color.
vshader_export_mask	wich of the last 6 ALU clauses is exporting
vshader_export_mode	0: position (1 vector), 1: position (2 vectors), 3:multipass
vshader_export_count[6]	# of interpolated parameters exported in clause 7 OR
	# of exported vectors to memory per clause in multipass mode (per clause)
kill_vector_on	use the mask kill vector to kill pixels and optimize texture pipe fetches OR use it as the fifth predicate vector wich is the only predicate vector kept across clause boundaries.
~~P_EXPORT[8]~~	~~number of exports for pixel shader~~
~~V_EXPORT[8]~~	~~number of exports for vertex shader for each clause. All numbers relate to the output buffer exports but for V_EXPORT[7] than can relate to the PC if Exports[7] is set to 00000.~~
~~ARBITRATION_policy~~	~~policy of the arbitration between vetexes and pixels~~
	~~Exports[8][6] Wich clause is exporting to the output buffer and what is it exporting.~~
	~~000000 : Not exporting (or exporting only to the PC)~~
	~~000001 : Exporting position (1)~~
	~~000010 : Exporting position (2)~~
	~~(1)00100 : Exporting RG~~

Formatted

~~(1)01000 : Exporting BA~~
~~(1)10000 : Exporting Z~~
~~If MSB set pixel shader exporting linear to memory not to Frame Buffer.~~
~~CST_SIZE_P Size of the constant store for pixels~~
~~CST_SIZE_V Size of the constant store for vertexes~~

~~19.~~21. DEBUG registers

Formatted: Bullets and Numbering

PROB_ADDR instruction address where the first problem occurred
PROB_COUNT number of problems encountered during the execution of the program

~~20.~~22. Interfaces

Formatted: Bullets and Numbering

~~20.1~~22.1 External Interfaces

~~20.1.1~~22.1.1 *PA/SC to ~~RE~~ SP0 : IJ bus*

This is a bus that sends the IJ information to the IJ fifos on the top of each shader pipe. At the same time the control information goes to the sequencer. There are 4 of these buses over the whole chip (SP0 thru 3)

Name	Direction	Bits	Description
IJs	PA→~~RE~~SP0	63	IJ information sent over 2 clocks
Mask	PA→~~RE~~SP0	1	Write Mask

~~20.1.2~~22.1.2 *PA/SC to SEQ : IJ Control bus*

Formatted: Bullets and Numbering

This is the control information sent to the sequencer in order to control the IJ fifos and all other information needed to execute a shader program on the sent pixels.

Name	Direction	Bits	Description
Write Mask	PA→SEQ(~~RE)~~SP)	4	Quad Write mask left to right
RB_ID	PA→SEQ(SP)~~PA→SEQ(RE)~~	8	RB id for each quad sent 2 bits per quad
LOD_CORRECT	PA→SEQ(SP)~~PA→SEQ(RE)~~	24	LOD correction per quad (6 bits per quad)
FVTX	PA→SEQ(SP)~~PA→SEQ(RE)~~	2	Provoking vertex for flat shading
PPTR0	PA→SEQ(SP)~~PA→SEQ(RE)~~	11	P Store pointer for vertex 0
PPRT1	PA→SEQ(SP)~~PA→SEQ(RE)~~	11	P Store pointer for vertex 1
PPTR2	PA→SEQ(SP)~~PA→SEQ(RE)~~	11	P Store pointer for vertex 2
E_OFF_VECTOR	PA→SEQ(SP)~~PA→SEQ(RE)~~	1	End of the vector
DEALLOC	PA→SEQ(SP)~~PA→SEQ(RE)~~	1	Deallocation token for the P Store
STATE	PA→SEQ(SP)~~PA→SEQ(RE)~~	21	State/constant pointer (6*3+3)
VALID	PA→SEQ(SP)~~PA→SEQ(RE)~~	16	Valid bits for all pixels
NULL	PA→SEQ(SP)~~PA→SEQ(RE)~~	1	Null Primitive (for PC deallocation purposes)
E_OFF_PRIM	PA→SEQ(SP)~~PA→SEQ(RE)~~	1	End Of the primitive
FBFACE	PA→SEQ(SP)~~PA→SEQ(RE~~	1	Front face = 1, back face = 0

	~~)~~		
TYPE	PA→SEQ(SP)~~PA→SEQ(RE)~~	3	Stippled line and Real time command need to load tex cords from alternate buffer 000 : Normal 001 : Stippled line 011 : Real Time 100 : Line AA 101 : Point AA 110 : Sprite
RTRn	SEQ→PA	1	Stalls the PA in n clocks
RTS	PA→SEQ(SP)~~PA→SEQ(RE)~~	1	PA ready to send data

~~20.1.3~~22.1.3 *VGT to ~~RE~~ SP : Vertex Bus*

Formatted: Bullets and Numbering

Name	**Direction**	**Bits**	**Description**
Vertex indexes	VGT→RE	128	Pointers of indexes or HOS surface information
EOF_vector	VGT→RE	1	End of the vector
Inputs_vert	VGT→RE	1	0: Normal 128 bits per vert 1: double 256 bits per vert
STATE	VGT→SEQ	21	Render State (6*3+3 for constants)

~~20.1.4~~22.1.4 *CP to SEQ : Constant store load*

Formatted: Bullets and Numbering

Name	**Direction**	**Bits**	**Description**
Constant Address	CP→SEQ	8	Address of the block of 4 constants
Constant Data	CP→SEQ	512	Data sent over 4 clocks
Remap Address	CP→SEQ	10	Remaping address write address
Remap Data pointer	CP→SEQ	8	Remaping pointer

~~20.1.5~~22.1.5 *CP to SEQ : Fetch State store load*

Formatted: Bullets and Numbering

Name	**Direction**	**Bits**	**Description**
Constant Address	CP→SEQ	8	Address of the block of 4 state constants
Constant Data	CP→SEQ	512	Data sent over 4 clocks
Remap Address	CP→SEQ	10	Remaping address write address
Remap Data pointer	CP→SEQ	8	Remaping pointer

~~20.1.6~~22.1.6 *CP to SEQ : Control State store load*

Formatted: Bullets and Numbering

Name	**Direction**	**Bits**	**Description**

{ISSUE: How,Who and what is the size of this bus?}

~~20.1.7~~22.1.7 *MH to SEQ: Instruction store Load*

Formatted: Bullets and Numbering

Name	**Direction**	**Bits**	**Description**
Instruction address	MH→SEQ	12	Instruction address
Instruction	MH→SEQ	96	Instruction X times
Control Instruction address	MH→SEQ	9	Pointer to the control instruction store
Control Instruction	MH→SEQ	32	Control Instruction X times

{ISSUE: CP or MH?}

~~20.1.8~~22.1.8 *SP to RB : Pixel read from RBs*

Formatted: Bullets and Numbering

Name	**Direction**	**Bits**	**Description**
Export_data	SP→RB	64*16	32~~a~~ pairs of 32 bits channel values
~~ExportID~~Shader_Dest	SP→RB	~~9~~4	Specifies one of the of up to 12 export destinations~~0cvvvvhqq: Vertex data vvvv 0-15 from first or second clause (c=0 or 1), XY or ZW components (h=0~~

			~~or 1), quad 0-3 in the shader (qq= 0-3)~~ ~~1cbbkttqq: Pixel data for buffer bb (0-3) from first or second clause (0-1) killed or not (k=1 or 0) quad 0-3 in the shader and data is RG (tt=0), BA (tt=1) or Z (tt=2)~~
Shader_Count~~ExportMask~~	SP→RB	~~2~~3	Each set of four pixels or vectors is exported over eight clocks. This field specifies where the SP is in that sequence.~~Specifies whether to write low, high or both 32 bit words. If export mask is 00 data is invalid~~
Shader_Last~~ExportLast~~	SP→RB	1	The current export clause is over (true for one clock) The last export instruction creates *two* cycles to the RB. This needs to be set on or after the last RB cycle that is produced by the last export instruction, but before the first RB cycle of the first export instruction of the next clause.~~Last export instruction of the clause~~
Shader_PixelValid	SP→RB	4x4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
Shader_WordValid	SP→RB	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors

~~20.1.9~~22.1.9 SEQ to RB : Control bus

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
Export_Pixel~~Type~~	SEQ→RB	1	1~~0~~: Pixel ~~1~~0: Vertex
Export_SEND~~Interleaving~~	SEQ→RB	1	Raised to indicate that the SQ is starting an export~~0: first interleaved clause~~ ~~1: second interleaved clause~~
Export_Clause~~Export_size~~	SEQ→RB	~~4~~3	Clause number, which is needed for vertex clauses~~0 thru 16 parameters exported for vertexes (vvvv) OR (bbzs) 1-4 color buffers (bb), two component (s=0) or 4 component colors (s=1) with z (z=1) or without z (z=0)~~
Export_State~~Valid~~	SEQ→RB	~~1~~21?	State ID, which is needed for vertex clauses~~Data valid~~

These fields are sent synchronously with SP export data, described in SP→RB interface

{ISSUE: Where are the PC pointers}~~Only one exporting clause (7) can be selected at any given time.~~

~~20.1.10~~22.1.10 RB to SEQ : Output file control

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
Export_RTS~~Buff_Full~~	RB→SEQ	1	Raised by RB to indicate that the following two fields reflect the result of the most recent export~~Set if full~~
Export_Position~~Avail_size~~	RB→SEQ	1~~6~~	Specifies whether there is room for another position.~~Size available in output buffers (in 32bits increments)~~
Export_Buffer	RB→SEQ	7	Specifies the space availble in the output buffers. 0: buffers are full 1: 2K-bits available (32-bits for each of the 64 pixels in a clause) ... 64: 128K-bits available (16 128-bit entries for each of 64 pixels) 65-127: RESERVED

~~20.1.11~~22.1.11 *SP to RB : Position return bus*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
Position return	SP→RB	128	Position data or sprite size (per clock)
Parameter cache pointer	SP→RB	11	Pointer where the data will be in the parameter cache for each vertex

For point sprites and position exports the size and position are interleaved on a 16 x 16 basis. We export 1 position then 1 point sprite sizes. The storage used is of 64x128 bits for position and 64x32 bits for sprite size, it is taken from the output buffer. Additionnally,if needed the edge flags are packed into the bits of the sprite sizes.

~~20.1.12~~22.1.12 *Shader Engine to Fetch Unit Bus (Fast Bus)*

Formatted: Bullets and Numbering

Four quad's worth of addresses is transferred to Fetch Unit every clock. These are sourced from a different pixel within each of the sub-engines repeating every 4 clocks. The register file index to read must precede the data by 2 clocks. The Read address associated with Quad 0 must be sent 1 clock after the Instruction Start signal is sent, so that data is read 3 clocks after the Instruction Start.

Four Quad's worth of Fetch Data may be written to the Register file every clock. These are directed to a different pixel of the sub-engines repeating every 4 clocks. The register file index to write must accompany the data. Data and Index associated with the Quad 0 must be sent 3 clocks after the Instruction Start signal is sent.

Name	Direction	Bits	Description
Tex_RegFile_Read_Data	SP->TEX	2048	16 Fetch Addresses read from the Register file
Tex_RegFile_Write_Data	TEX→SP	2048	16 texture results

Formatted

~~20.1.13~~22.1.13 *Sequencer to Fetch Unit bus (Slow Bus)*

Formatted: Bullets and Numbering

Once every four clock, the fetch unit sends to the sequencer on wich clause it is now working and if the data in the registers is ready or not. This way the sequencer can update the fetch counters for the reservation station fifos. The sequencer also provides the intruction and constants for the fetch to execute and the address in the register file where to write the fetch return data.

Name	Direction	Bits	Description
Tex_Ready	TEX→ SEQ	1	Data ready
Tex_Clause_Num	TEX→ SEQ	3	Clause number
Tex_cst	SEQ→TEX	10	Fetch state address 10 bits sent over 4 clocks
Tex_Inst	SEQ→TEX	12	Fetch instruction address 12 bits sent over 4 clocks
EO_CLAUSE	SEQ→TEX	1	Last instruction of the clause
PHASE	SEQ→TEX	1	Write phase signal
LOD_CORRECT	SEQ→TEX	96	LOD correct 3 bits per comp 2 components per quad * 16 quads
Mask	SEQ→TEX	64	Pixel mask 1 bit per pixel
Tex_Clause_Num	SEQ→TEX	3	Clause number
Tex_Write_Register_Index	SEQ->TEX	7	Index into Register file for write of returned Fetch Data
Tex_Read_Register_Index	SEQ->SP	7	Index into Register files for reading Fetch Address (internal)

~~21.~~23. Internal interfaces

Formatted: Bullets and Numbering

~~21.1.1~~23.1.1 *RE to SEQ : Vertex Control Bus*

Name	Direction	Bits	Description
STATE	VGT→SEQ	21	Render State (6*3+3 for constants)
Vert counter	VGT→SEQ	6	Which vertices are valid
Inputs_vert	VGT→SEQ	1	0: Normal 128 bits per vert 1: double 256 bits per vert

This information is sent over 4 clocks.~~s information needs to be sent over 64 clocks.~~

Formatted: Bullets and Numbering

~~22.~~24. Examples of program executions

~~22.1.1~~24.1.1 *Sequencer Control of a Vector of Vertices*

1. PA sends a vector of 64 vertices (actually vertex indices – 32 bits/index for 2048 bit total) to the RE's Vertex FIFO
   - state pointer as well as tag into position cache is sent along with vertices
   - space was allocated in the position cache for transformed position before the vector was sent
   - **also before the vector is sent to the RE, the CP has loaded the global instruction store with the vertex shader program (using the MH?)**
   - The vertex program is assumed to be loaded when we receive the vertex vector.
     - the SEQ then accesses the IS base for this shader using the local state pointer (provided to all sequencers by the RBBM when the CP is done loading the program)
2. SEQ arbitrates between the Pixel FIFO and the Vertex FIFO – basically the Vertex FIFO always has priority
   - at this point the vector is removed from the Vertex FIFO
   - the arbitrer is not going to select a vector to be transformed if the parameter cache is full unless the pipe as nothing else to do (ie no pixels are in the pixel fifo).
3. SEQ allocates space in the SP register file for index data plus GPRs used by the program
   - the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer that came down with the vertices
   - SEQ will not send vertex data until space in the register file has been allocated
4. SEQ sends the vector to the SP register file over the RE_SP interface (which has a bandwidth of 2048 bits/cycle)
   - the 64 vertex indices are sent to the 64 register files over 4 cycles
     - RF0 of SU0, SU1, SU2, and SU3 is written the first cycle
     - RF1 of SU0, SU1, SU2, and SU3 is written the second cycle
     - RF2 of SU0, SU1, SU2, and SU3 is written the third cycle
     - RF3 of SU0, SU1, SU2, and SU3 is written the fourth cycle
   - the index is written to the least significant 32 bits **(floating point format?) (what about compound indices)** of the 128-bit location within the register file (w); the remaining data bits are set to zero (x, y, z)
5. SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of fetch state machine 0, or TSM0 FIFO)
   - the control packet contains the state pointer, the tag to the position cache and a register file base pointer.
6. TSM0 accepts the control packet and fetches the instructions for fetch clause 0 from the global instruction store
   - TSM0 was first selected by the TSM arbiter before it could start
7. all instructions of fetch clause 0 are issued by TSM0
8. the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)
   - TSM0 does not wait for requests made to the Fetch Unit to complete; it passes the register file write index for the fetch data to the TU, which will write the data to the RF as it is received
   - once the TU has written all the data to the register files, it increments a counter that is associated with ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go ahead start to execute the ALU clause
9. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store
10. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of fetch state machine 1, or TSM1 FIFO)
11. the control packet continues to travel down the path of reservation stations until all clauses have been executed
    - position can be exported in ALU clause 3 (or 4?); the data (and the tag) is sent over a position bus (which is shared with all four shader pipes) back to the PA's position cache
    - A parameter cache pointer is also sent along with the position data. This tells to the PA where the data is going to be in the parameter cache.
      - there is a position export FIFO in the SP that buffers position data before it gets sent back to the PA

    - the ASM arbiter will prevent a packet from starting an exporting clause if the position export FIFO is full
- parameter data is exported in clause 7 (as well as position data if it was not exported earlier)
    - parameter data is sent to the Parameter Cache over a dedicated bus
    - the SEQ allocates storage in the Parameter Cache, and the SEQ deallocates that space when there is no longer a need for the parameters (it is told by the PA when using a token).
    - the ASM arbiter will prevent a packet from starting on ASM7 if the parameter cache (or the position buffer if position is being exported) is full

12. after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program

~~22.1.2~~24.1.2 *Sequencer Control of a Vector of Pixels*

Formatted: Bullets and Numbering

1. **As with vertex shader programs, pixel shaders are loaded into the global instruction store by the CP**
    - At this point it is assumed that the pixel program is loaded into the instruction store and thus ready to be read.
2. the RE's Pixel FIFO is loaded with the barycentric coordinates for pixel quads by the detailed walker
    - the state pointer and the LOD correction bits are also placed in the Pixel FIF0
    - the Pixel FIFO is wide enough to source four quad's worth of barycentrics per cycle
3. SEQ arbitrates between Pixel FIFO and Vertex FIFO – when there are no vertices pending OR there is no space left in the register files for vertices, the Pixel FIFO is selected
4. SEQ allocates space in the SP register file for all the GPRs used by the program
    - the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer
    - SEQ will not allow interpolated data to be sent to the shader until space in the register file has been allocated
5. SEQ controls the transfer of interpolated data to the SP register file over the RE_SP interface (which has a bandwidth of 2048 bits/cycle). See interpolated data bus diagrams for details.
6. SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of fetch state machine 0, or TSM0 FIFO)
    - note that there is a separate set of reservation stations/arbiters/state machines for vertices and for pixels
    - the control packet contains the state pointer, the register file base pointer, and the LOD correction bits
    - all other informations (such as quad address for example) travels in a separate FIFO
7. TSM0 accepts the control packet and fetches the instructions for fetch clause 0 from the global instruction store
    - TSM0 was first selected by the TSM arbiter before it could start
8. all instructions of fetch clause 0 are issued by TSM0
9. the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)
    - TSM0 does not wait for fetch requests made to the Fetch Unit to complete; it passes the register file write index for the fetch data to the TU, which will write the data to the RF as it is received
    - once the TU has written all the data for a particular clause to the register files, it increments a counter that is associated with the ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go ahead and pop the FIFO and start to execute the ALU clause
10. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store
11. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of fetch state machine 1, or TSM1 FIFO)
12. the control packet continues to travel down the path of reservation stations until all clauses have been executed
    - pixel data is exported in the last ALU clause (clause 7)
        - it is sent to an output FIFO where it will be picked up by the render backend
        - the ASM arbiter will prevent a packet from starting on ASM7 if the output FIFO is full
13. after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program

~~22.1.3~~24.1.3 *Notes*

Formatted: Bullets and Numbering

14. T~~t~~he state machines and arbitrers will operate ahead of time so that they will be able to immediately start the real threads or stall.
15. T~~t~~he register file base pointer for a vector needs to travel with the vector through the reservation stations, but the instruction store base pointer does not – this is because the RF pointer is different for all threads, but the IS pointer is only different for each state and thus can be accessed via the state pointer
16. Waterfalling~~, parameter buffer allocation, loops and branches and parameter cache de-allocation~~ still needs to be specked out.

~~23.~~25. Open issues

Formatted: Bullets and Numbering

There is currently an issue with constants. If the constants are not the same for the whole vector of vertices, we don’t have the bandwith from the fetch store to feed the ALUs. Two solutions exists for this problem:

1) Let the compiler handle the case and put those instructions in a fetch clause so we can use the bandwith there to operate. This requires a significant amount of temporary storage in the register store.
2) Waterfall down the pipe allowing only at a given time the vertices having the same constants to operate in parrallel. This might in the worst case slow us down by a factor of 16.

Need to do some testing on the size of the register file as well as on the register file allocation method (dynamic VS static).

Saving power?

~~Size of the fifo containing the information of a vector of pixels/vertices. And size of the fifos before the reservation stations.~~

~~Loops and branches.~~

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 2015~~5~~ November, 200126	GEN-CXXXXX-REVA	1 of 32

Author: Laurent Lefebvre

Issue To:	**Copy No:**

R400 Sequencer Specification

SEQ

Version 1.1~~0~~

Overview: This is an architectural specification for the R400 Sequencer block (SEQ). It provides an overview of the required capabilities and expected uses of the block. It also describes the block interfaces, internal sub-blocks, and provides internal state diagrams.

AUTOMATICALLY UPDATED FIELDS:
Document Location: C:\perforce\r400\arch\doc\gfx\RE\R400_Sequencer.doc
Current Intranet Search Title: R400 Sequencer Specification

APPROVALS	
Name/Dept	Signature/Date

Remarks:

THIS DOCUMENT CONTAINS CONFIDENTIAL INFORMATION THAT COULD BE SUBSTANTIALLY DETRIMENTAL TO THE INTEREST OF ATI TECHNOLOGIES INC. THROUGH UNAUTHORIZED USE OR DISCLOSURE.

Table Of Contents

1. OVERVIEW .... 5~~4~~
1.1 Top Level Block Diagram .... 7~~6~~
1.2 Data Flow graph .... 9~~8~~
1.3 Control Graph .... 10~~9~~
2. INTERPOLATED DATA BUS .... 10~~9~~
3. INSTRUCTION STORE .... 13~~12~~
4. SEQUENCER INSTRUCTIONS .... 13~~12~~
5. CONSTANT STORE .... 13~~12~~
6. LOOPING AND BRANCHES .... 13~~12~~
6.1 The controlling state .... 13~~12~~
6.2 The Control Flow Program .... 14~~13~~
6.3 Data dependant predicate instructions .... 17~~15~~
6.4 HW Detection of PV,PS .... 18~~16~~
6.5 Register file indexing .... 18~~16~~
6.6 Predicated Instruction support for Texture clauses .... 18~~16~~
6.7 Debugging the Shaders .... 18~~16~~
6.7.1 Method 1: Debugging registers .... 18~~16~~
6.7.2 Method 2: Exporting the values in the GPRs (12) 19~~17~~
6.7.3 Method 3: Selective export of a 32 bit Dword. 19~~17~~
7. PIXEL KILL MASK .... 20~~18~~
8. HOS SURFACES .... 20~~18~~
9. REGISTER FILE ALLOCATION .... 20~~18~~
10. FETCH ARBITRATION .... 22~~19~~
11. ALU ARBITRATION .... 22~~19~~
12. HANDLING STALLS .... 23~~20~~
13. CONTENT OF THE RESERVATION STATION FIFOS 23~~20~~
14. THE OUTPUT FILE .... 23~~20~~
15. IJ FORMAT .... 23~~20~~
16. THE PARAMETER CACHE .... 24~~21~~
17. VERTEX POSITION EXPORTING .... 24~~21~~
18. EXPORTING ARBITRATION .... 24~~21~~
19. REAL TIME COMMANDS .... 24~~21~~
20. REGISTERS .... 25~~22~~
20.1 Control .... 25~~22~~
20.2 Context .... 25~~22~~
21. DEBUG REGISTERS .... 25~~22~~
21.1 Control .... 25~~22~~
21.2 Context .... 26~~22~~
22. INTERFACES .... 26~~23~~
22.1 External Interfaces .... 26~~23~~
22.1.1 PA/SC to SP0 : IJ bus .... 26~~23~~
22.1.2 PA/SC to SEQ : IJ Control bus .... 26~~23~~
22.1.3 SEQ to SP0 : Interpolator bus .... 26~~23~~
22.1.4 SEQ to SP0 : Parameter Cache bus 27~~24~~
22.1.5 SEQ to SX0 : Parameter Cache Mux control Bus 27~~24~~
22.1.6 SX0 to SP0 : Parameter Cache Return bus 27~~24~~
22.1.7 VGT to SP0/SEQ : Vertex Bus .... 27~~24~~
22.1.8 CP to SEQ : Constant store load .. 27~~24~~
22.1.9 CP to SEQ : Fetch State store load 27~~24~~
22.1.10 CP to SEQ : Control State store load 28~~24~~
22.1.11 MH to SEQ: Instruction store Load 28~~25~~
22.1.12 SP0 to SX0 : Pixel read from RBs 28~~25~~
22.1.13 SEQ to SX0 : Control bus .... 28~~25~~
22.1.14 SX0 to SEQ : Output file control ... 28~~25~~
22.1.15 SP0 to SX0 : Position return bus .. 29~~26~~
22.1.16 Shader Engine to Fetch Unit Bus (Fast Bus) 29~~26~~
22.1.17 Sequencer to Fetch Unit bus (Slow Bus) 29~~26~~
23. EXAMPLES OF PROGRAM EXECUTIONS 30~~26~~
23.1.1 Sequencer Control of a Vector of Vertices 30~~26~~
23.1.2 Sequencer Control of a Vector of Pixels 31~~27~~
23.1.3 Notes .... 32~~28~~
24. OPEN ISSUES .... 32~~28~~
~~1. OVERVIEW .... 4~~
~~1.1 Top Level Block Diagram .... 6~~
~~1.2 Data Flow graph .... 8~~
~~1.3 Control Graph .... 9~~
~~2. INTERPOLATED DATA BUS .... 9~~
~~3. INSTRUCTION STORE .... 12~~
~~4. SEQUENCER INSTRUCTIONS .... 12~~
~~5. CONSTANT STORE .... 12~~
~~6. LOOPING AND BRANCHES .... 12~~
~~6.1 The controlling state .... 12~~
~~6.2 The Control Flow Program .... 13~~
~~6.3 Data dependant predicate instructions .... 16~~
~~6.4 Register file indexing .... 16~~
~~7. PIXEL KILL MASK .... 17~~
~~8. HOS SURFACES .... 17~~
~~9. REGISTER FILE ALLOCATION .... 17~~
~~10. FETCH ARBITRATION .... 18~~
~~11. ALU ARBITRATION .... 18~~
~~12. HANDLING STALLS .... 19~~
~~13. CONTENT OF THE RESERVATION STATION FIFOS 19~~
~~14. THE OUTPUT FILE .... 19~~
~~15. IJ FORMAT .... 19~~
~~16. THE PARAMETER CACHE .... 20~~
~~17. VERTEX POSITION EXPORTING .... 20~~
~~18. EXPORTING ARBITRATION .... 20~~
~~19. REAL TIME COMMANDS .... 20~~
~~20. REGISTERS .... 20~~
~~20.1 Control .... 20~~
~~20.2 Context .... 21~~
~~21. DEBUG REGISTERS .... 21~~

~~22. INTERFACES ....21~~
~~22.1 External Interfaces ....21~~
~~22.1.1 PA/SC to SP0 : IJ bus ....21~~
~~22.1.2 PA/SC to SEQ : IJ Control bus....22~~
~~22.1.3 VGT to SP : Vertex Bus....22~~
~~22.1.4 CP to SEQ : Constant store load ....22~~
~~22.1.5 CP to SEQ : Fetch State store load .22~~
~~22.1.6 CP to SEQ : Control State store load 23~~
~~22.1.7 MH to SEQ: Instruction store Load ..23~~
~~22.1.8 SP to RB : Pixel read from RBs....23~~
~~22.1.9 SEQ to RB : Control bus ....23~~
~~22.1.10 RB to SEQ : Output file control....23~~
~~22.1.11 SP to RB : Position return bus....24~~
~~22.1.12 Shader Engine to Fetch Unit Bus (Fast Bus) 24~~
~~22.1.13 Sequencer to Fetch Unit bus (Slow Bus) 24~~
~~23. INTERNAL INTERFACES....24~~
~~23.1.1 RE to SEQ : Vertex Control Bus....24~~
~~24. EXAMPLES OF PROGRAM EXECUTIONS..25~~
~~24.1.1 Sequencer Control of a Vector of Vertices 25~~
~~24.1.2 Sequencer Control of a Vector of Pixels 26~~
~~24.1.3 Notes ....27~~
~~25. OPEN ISSUES....27~~
~~1. OVERVIEW....4~~
~~1.1 Top Level Block Diagram....6~~
~~1.2 Data Flow graph....8~~
~~1.3 Control Graph ....9~~
~~2. INTERPOLATED DATA BUS ....9~~
~~3. INSTRUCTION STORE....12~~
~~4. SEQUENCER INSTRUCTIONS....12~~
~~5. CONSTANT STORE....12~~
~~6. LOOPING AND BRANCHES....12~~
~~6.1 The controlling state....12~~
~~6.2 The Control Flow Program....13~~
~~6.3 Data dependant predicate instructions....16~~
~~6.4 Register file indexing....16~~
~~7. HOS SURFACES ....17~~
~~8. REGISTER FILE ALLOCATION....17~~
~~9. FETCH ARBITRATION ....18~~
~~10. ALU ARBITRATION....18~~
~~11. HANDLING STALLS....19~~
~~12. CONTENT OF THE RESERVATION STATION FIFOS19~~
~~13. THE OUTPUT FILE ....19~~
~~14. IJ FORMAT....19~~
~~15. THE PARAMETER CACHE ....20~~
~~16. VERTEX POSITION EXPORTING ....20~~
~~17. REAL TIME COMMANDS....20~~
~~18. REGISTERS ....20~~
~~18.1 Control....20~~
~~18.2 Context ....21~~
~~19. DEBUG REGISTERS ....21~~
~~20. INTERFACES....21~~
~~20.1 External Interfaces ....21~~
~~20.1.1 PA/SC to RE : IJ bus ....21~~
~~20.1.2 PA/SC to SEQ : IJ Control bus....21~~
~~20.1.3 VGT to RE : Vertex Bus ....22~~
~~20.1.4 CP to SEQ : Constant store load....22~~
~~20.1.5 CP to SEQ : Fetch State store load..22~~
~~20.1.6 CP to SEQ : Control State store load 22~~
~~20.1.7 MH to SEQ: Instruction store Load ..22~~
~~20.1.8 SP to RB : Pixel read from RBs....23~~
~~20.1.9 SEQ to RB : Control bus ....23~~
~~20.1.10 RB to SEQ : Output file control....23~~
~~20.1.11 SP to RB : Position return bus....23~~
~~20.1.12 Shader Engine to Fetch Unit Bus (Fast Bus) 24~~
~~20.1.13 Sequencer to Fetch Unit bus (Slow Bus) 24~~
~~21. INTERNAL INTERFACES....24~~
~~21.1.1 RE to SEQ : Vertex Control Bus....24~~
~~22. EXAMPLES OF PROGRAM EXECUTIONS..24~~
~~22.1.1 Sequencer Control of a Vector of Vertices 24~~
~~22.1.2 Sequencer Control of a Vector of Pixels 25~~
~~22.1.3 Notes ....26~~
~~23. OPEN ISSUES....26~~

Revision Changes:

Rev 0.1 (Laurent Lefebvre) Date: May 7, 2001	First draft.
Rev 0.2 (Laurent Lefebvre) Date : July 9, 2001	Changed the interfaces to reflect the changes in the SP. Added some details in the arbitration section.
Rev 0.3 (Laurent Lefebvre) Date : August 6, 2001	Reviewed the Sequencer spec after the meeting on August 3, 2001.
Rev 0.4 (Laurent Lefebvre) Date : August 24, 2001	Added the dynamic allocation method for register file and an example (written in part by Vic) of the flow of pixels/vertices in the sequencer.
Rev 0.5 (Laurent Lefebvre) Date : September 7, 2001	Added timing diagrams (Vic)

Rev 0.6 (Laurent Lefebvre) Date : September 24, 2001	Changed the spec to reflect the new R400 architecture. Added interfaces.
Rev 0.7 (Laurent Lefebvre) Date : October 5, 2001	Added constant store management, instruction store management, control flow management and data dependant predication.
Rev 0.8 (Laurent Lefebvre) Date : October 8, 2001	Changed the control flow method to be more flexible. Also updated the external interfaces.
Rev 0.9 (Laurent Lefebvre) Date : October 17, 2001	Incorporated changes made in the 10/18/01 control flow meeting. Added a NOP instruction, removed the conditional_execute_or_jump. Added debug registers.
Rev 1.0 (Laurent Lefebvre) Date : October 19, 2001	Refined interfaces to RB. Added state registers.
Rev 1.1 (Laurent Lefebvre) Date : October 26, 2001	Added SEQ→SP0 interfaces. Changed delta precision. Changed VGT→SP0 interface. Debug Methods added.

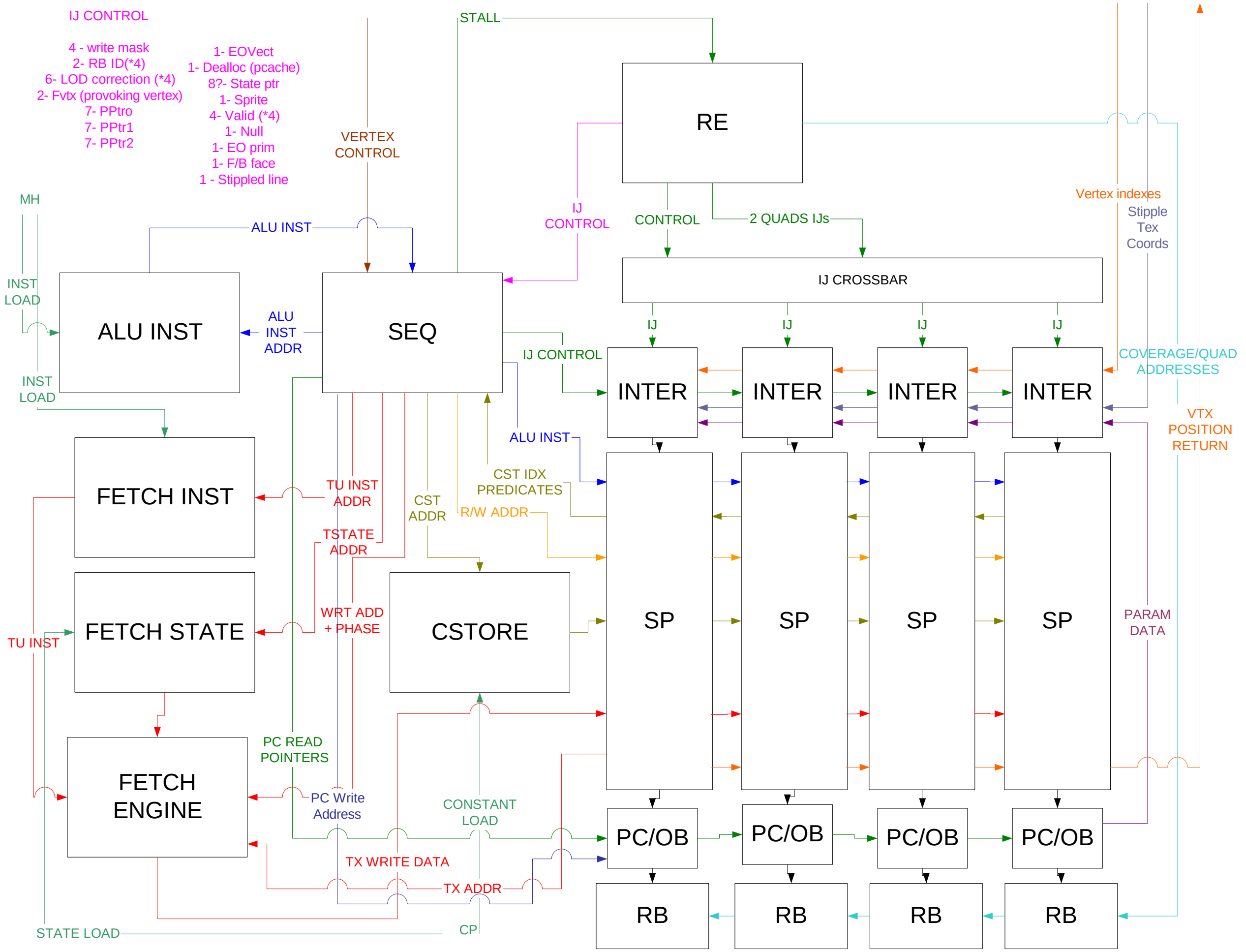
1. Overview

The sequencer first arbitrates between vectors of 64 vertices that arrive directly from primitive assembly and vectors of 16 quads (64 pixels) that are generated in the raster engine.

The vertex or pixel program specifies how many GPR's it needs to execute. The sequencer will not start the next vector until the needed space is available.

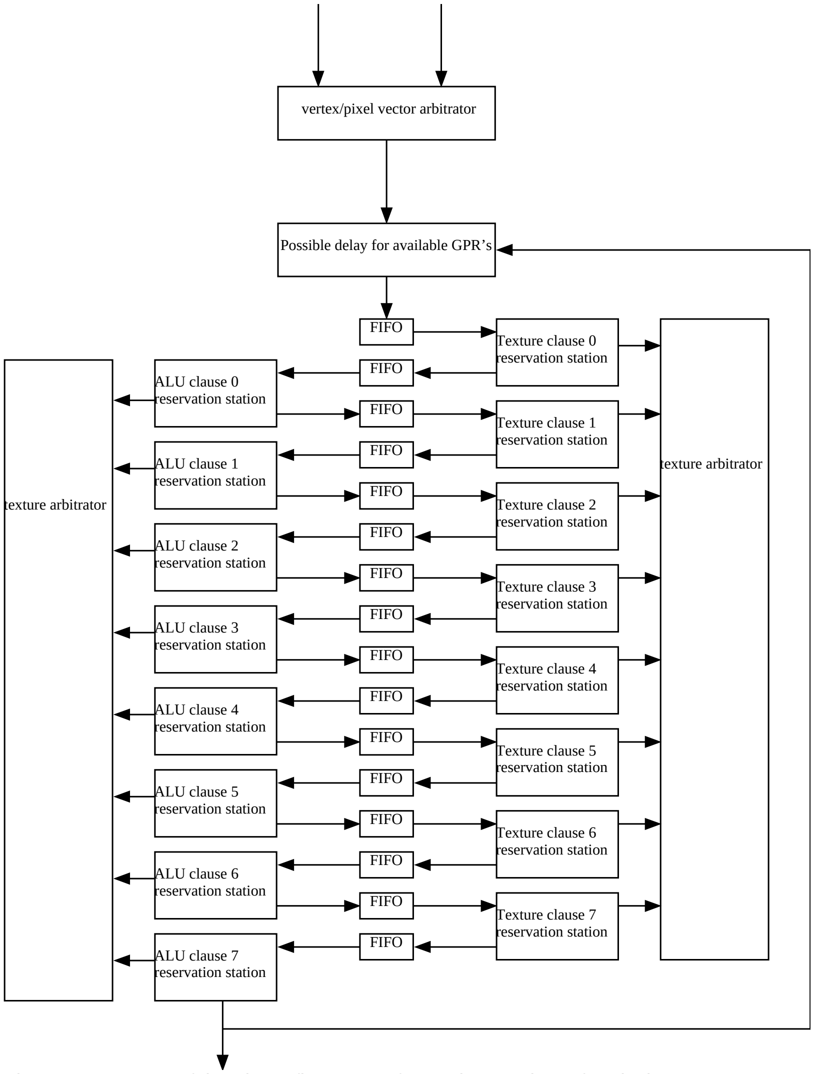
The sequencer is based on the R300 design. It chooses two ALU clauses and a fetch clause to execute, and executes all of the instructions in a clause before looking for a new clause of the same type. Two ALU clauses are executed interleaved to hide the ALU latency. Each vector will have eight fetch and eight ALU clauses, but clauses do not need to contain instructions. A vector of pixels or vertices ping-pongs along the sequencer FIFO, bouncing from fetch reservation station to alu reservation station. A FIFO exists between each reservation stage, holding up vectors until the vector currently occupying a reservation station has left. A vector at a reservation station can be chosen to execute. The sequencer looks at all eight alu reservation stations to choose an alu clause to execute and all eight fetch stations to choose a fetch clause to execute. The arbitrator will give priority to clauses/reservation stations closer to the bottom of the pipeline. It will not execute an alu clause until the fetch fetches initiated by the previous fetch clause have completed. There are two separate sets of reservation stations, one for pixel vectors and one for vertices vectors. This way a pixel can pass a vertex and a vertex can pass a pixel.

To support the shader pipe the raster engine also contains the shader instruction cache and constant store. There are only one constant store for the whole chip and one instruction store. These will be shared among the four shader pipes. The four shader pipes also execute the same instuction thus there is only one sequencer for the whole chip.



IJ CONTROL
4 - write mask
2- RB ID(*4)
6- LOD correction (*4)
2- Fvtx (provoking vertex)
7- PPtro
7- PPtr1
7- PPtr2
1- EOVect
1- Dealloc (pcache)
8?- State ptr
1- Sprite
4- Valid (*4)
1- Null
1- EO prim
1- F/B face
1 - Stippled line
VERTEX CONTROL
STALL
RE
IJ CONTROL
CONTROL
2 QUADS IJs
MH
ALU INST
INST LOAD
ALU INST
SEQ
ALU INST ADDR
INST LOAD
IJ CROSSBAR
IJ
IJ
IJ
IJ
IJ CONTROL
INTER
INTER
INTER
INTER
Vertex indexes
Stipple Tex Coords
COVERAGE/QUAD ADDRESSES
VTX POSITION RETURN
ALU INST
FETCH INST
TU INST ADDR
CST ADDR
CST IDX PREDICATES
R/W ADDR
TSTATE ADDR
FETCH STATE
WRT ADD + PHASE
CSTORE
TU INST
SP
SP
SP
SP
PARAM DATA
FETCH ENGINE
PC READ POINTERS
PC Write Address
CONSTANT LOAD
TX WRITE DATA
TX ADDR
PC/OB
PC/OB
PC/OB
PC/OB
RB
RB
RB
RB
STATE LOAD
CP

1.1 Top Level Block Diagram



vertex/pixel vector arbitrator
Possible delay for available GPR's
FIFO
Texture clause 0 reservation station
Texture clause 1 reservation station
Texture clause 2 reservation station
Texture clause 3 reservation station
Texture clause 4 reservation station
Texture clause 5 reservation station
Texture clause 6 reservation station
Texture clause 7 reservation station
texture arbitrator
ALU clause 0 reservation station
ALU clause 1 reservation station
ALU clause 2 reservation station
ALU clause 3 reservation station
ALU clause 4 reservation station
ALU clause 5 reservation station
ALU clause 6 reservation station
ALU clause 7 reservation station
texture arbitrator

There are two sets of the above figure, one for vertices and one for pixels.

Depending on the arbitration state, the sequencer will either choose a vertex or a pixel packet. The control packet consists of 21 bits of state, 6-7 bits for the base address of the Shader program and some information on the coverage to determine fetch LOD plus other various small state bits.

On receipt of a packet, the input state machine (not pictured but just before the first FIFO) allocated enough space in the registers to store the interpolated values and temporaries. Following this, the input state machine stacks the packet in the first FIFO.

On receipt of a command, the level 0 fetch machine issues a texure request and corresponding register address for the fetch address (ta). A small command (tcmd) is passed to the fetch system identifying the current level number (0) as well as the register write address for the fetch return data. One fetch request is sent every 4 clocks causing the texturing of sixteen 2x2s worth of data (or 64 vertices). Once all the requests are sent the packet is put in FIFO 1.

Upon recept of the return data, the fetch unit writes the data to the register file using the write address that was provided by the level 0 fetch machine and sends the clause number (0) to the level 0 fetch state machine to signify that the write is done and thus the data is ready. Then, the level 0 fetch machine increments the counter of FIFO 1 to signify to the ALU 1 that the data is ready to be processed.

On receipt of a command, the level 0 ALU machine first decrements the input FIFO counter and then issues a complete set of level 0 shader instructions. For each instruction, the state machine generates 3 source addresses, one destination address (3 cycles later) and an instruction. Once the last instruction as been issued, the packet is put into FIFO 2.

There will always be two active ALU clauses at any given time (and two arbitrers). One arbitrer will arbitrate over the odd instructions (4 clocks cycles) and the other one will arbitrate over the even instructions (4 clocks cycles). The only constraints between the two arbitrers is that they are not allowed to pick the same clause number as the other one is currently working on if the packet is not of the same type (render state).

If the packet is a vertex packet, upon reaching ALU clause 3, it can export the position if the position is ready. So the arbitrer must prevent ALU clause 3 to be selected if the positional buffer is full (or can’t be accessed). Along with the positional data, the location where the vertex data is to be put is also sent (parameter data pointers).

{ISSUE: How do we handle parameter cache pointers (computed, semi-computed or not computed)?}

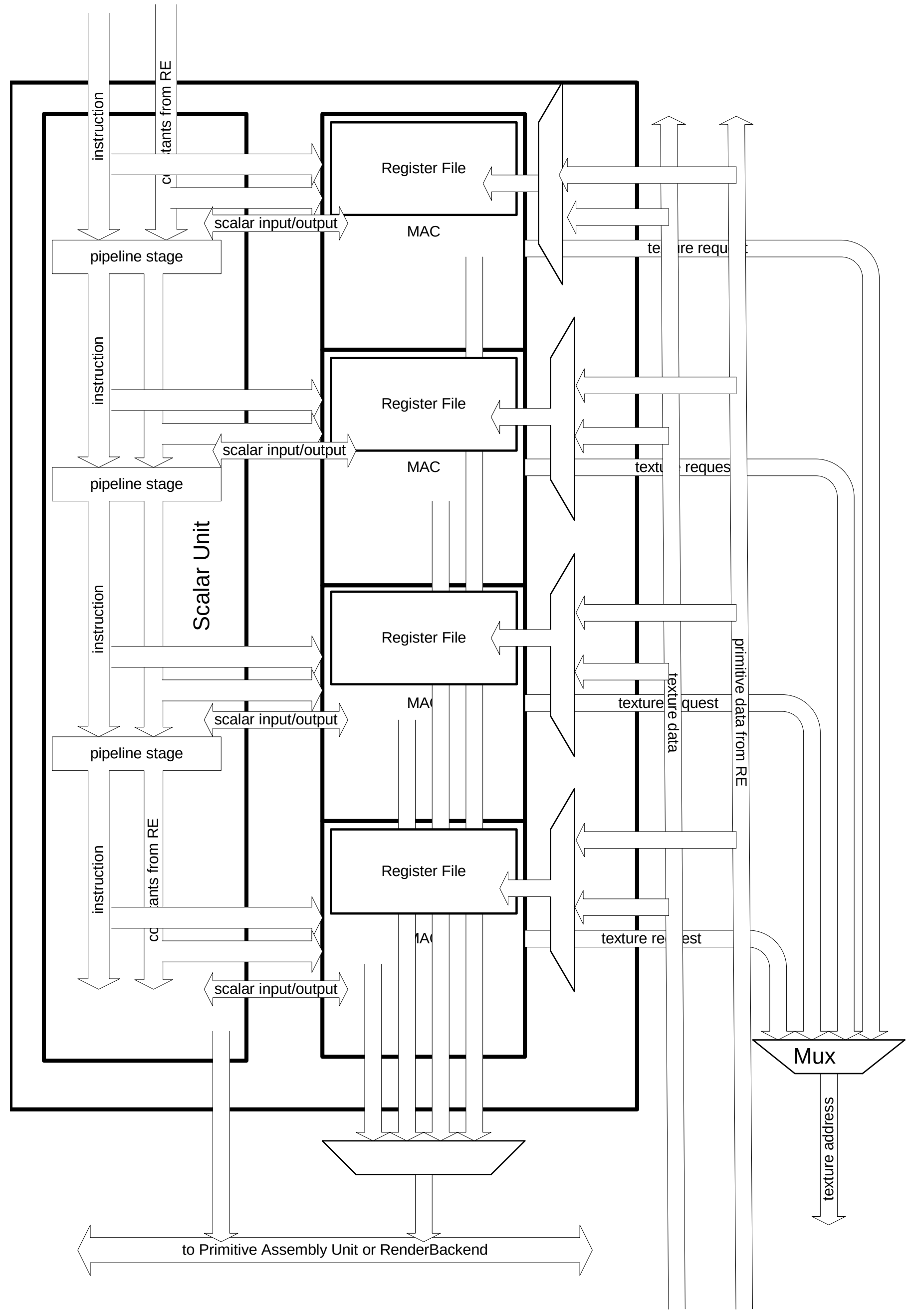
A special case is for HOS surfaces wich can export 12 parameters per last 6 clauses to the output buffer. If the output buffer is full or doesn’t have enough space the sequencer will prevent such a vertex group to enter an exporting clause.

Regular pixel and vertex shaders can export 12 parameters to memory from the last clause only (7).

All other level process in the same way until the packet finally reaches the last ALU machine (7). On completion of the level 7 ALU clause, a valid bit is sent to the Render Backend which picks up the color data. This requires that the last instruction writes to the output register – a condition that is almost always true. If the packet was a vertex packet, instead of sending the valid bit to the RB, it is sent to the PA so it can know that the data present in the parameter store is valid.

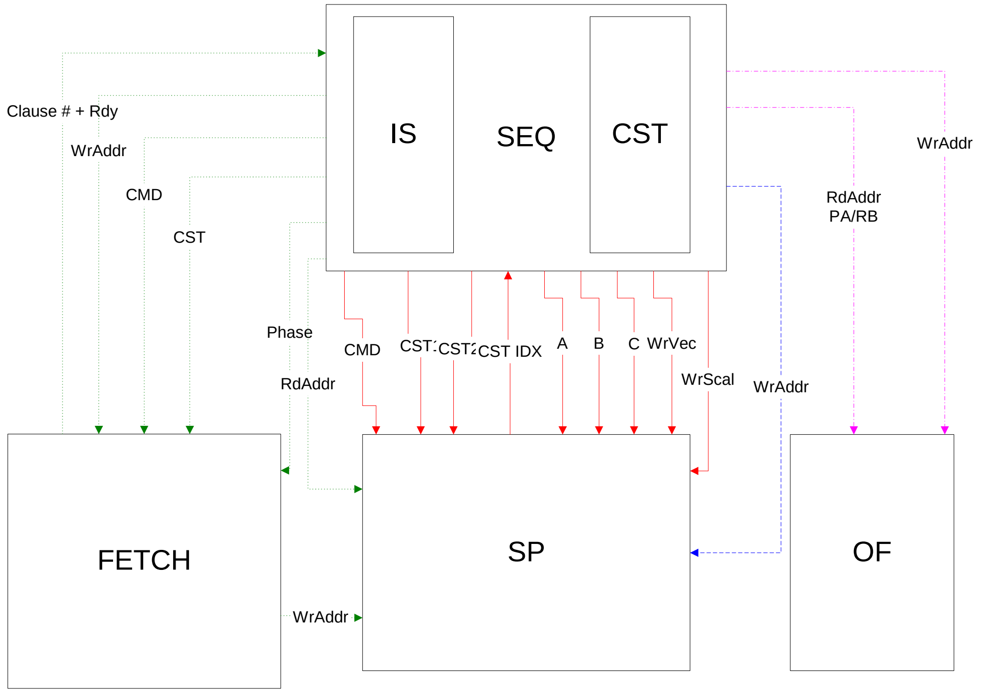
Only two ALU state machine may have access to the register file address bus or the instruction decode bus at one time. Similarly, only one fetch state machine may have access to the register file address bus at one time. Arbitration is performed by three arbitrer blocks (two for the ALU state machines and one for the fetch state machines). The arbitrers always favor the higher number state machines, preventing a bunch of half finished jobs from clogging up the register files.

1.2 Data Flow graph



The gray area represents blocks that are replicated 4 times per shader pipe (16 times on the overall chip).

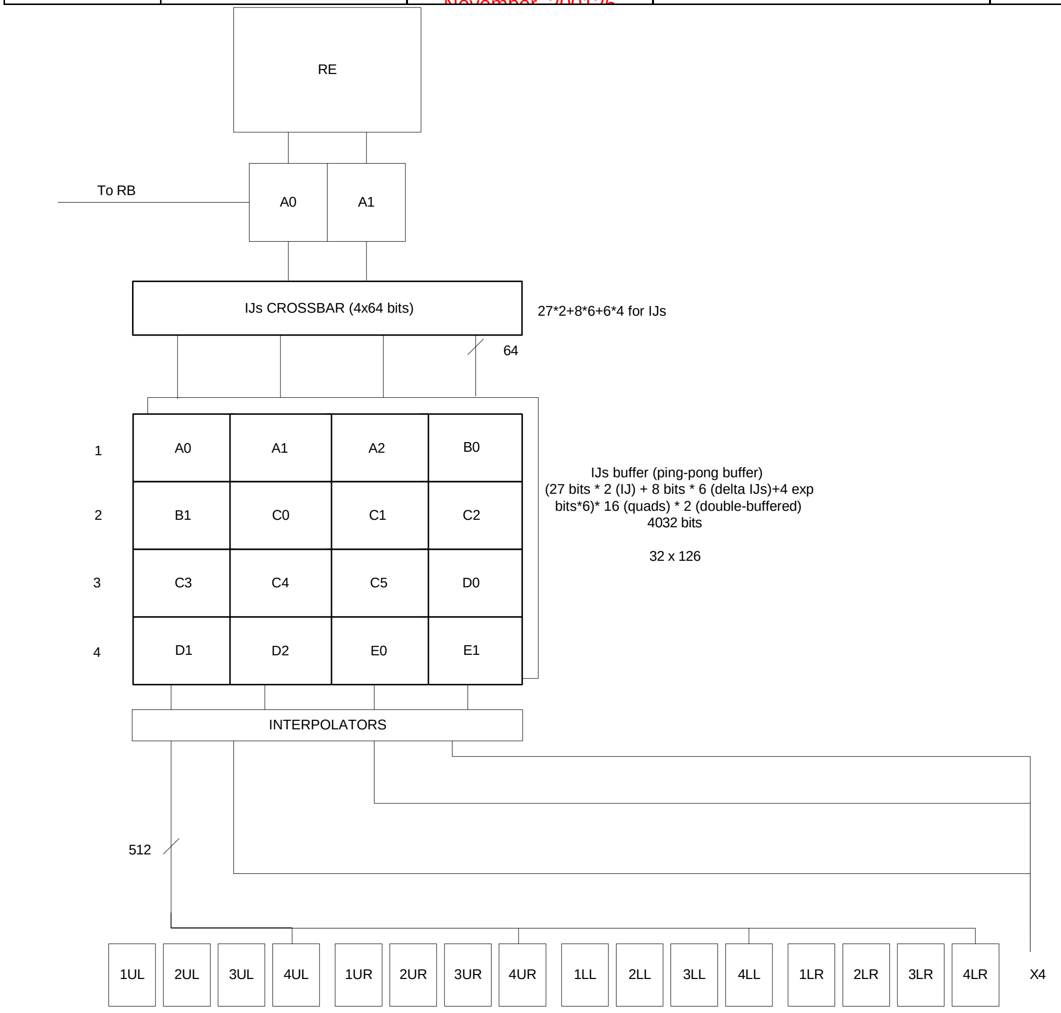
1.3 Control Graph



In green is represented the Fetch control interface, in red the ALU control interface, in blue the Interpolated/Vector control interface and in purple is the output file control interface.

2. Interpolated data bus

The interpolators contain an IJ buffer to pack the information as much as possible before writing it to the register file.



RE
To RB
A0
A1
IJs CROSSBAR (4x64 bits)
27*2+8*6+6*4 for IJs
64
1
2
3
4
A0
A1
A2
B0
B1
C0
C1
C2
C3
C4
C5
D0
D1
D2
E0
E1
IJs buffer (ping-pong buffer)
(27 bits * 2 (IJ) + 8 bits * 6 (delta IJs)+4 exp bits*6)* 16 (quads) * 2 (double-buffered)
4032 bits
32 x 126
INTERPOLATORS
512
1UL
2UL
3UL
4UL
1UR
2UR
3UR
4UR
1LL
2LL
3LL
4LL
1LR
2LR
3LR
4LR
X4

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19
SP0	A0	B1	C3	D1					A0	B1	C3	D1					V 0-3	V 16-19	V 32-35	V 48-51
SP1	A1		C4	D2		C0			A1		C4	D2		C0			V 4-7	V 20-23	V 36-39	V 52-55
SP2	A2		C5			C1		E0	A2		C5			C1		E0	V 8-11	V 24-27	V 40-43	V 56-59
SP3					B0	C2	D0	E1					B0	C2	D0	E1	V 12-15	V 28-31	V 44-47	V 60-63

P0

P1

Above is an example of a tile we might receive. The IJ information is packed in the IJ buffer 2 quads at a time. The sequencer allows at any given time as many as four quads to interpolate a parameter. They all have to come from the same primitive. Then the sequencer controls the write mask to the register to write the valid data in.

3. Instruction Store

There is going to be only one instruction store for the whole chip. It will contain 4096 instructions of 96 bits each. There is also going to be a control instruction store of size 256(512?)x32.

{ISSUE : The instruction store is loaded by the sequencer using the memory hub ?}.

The read bandwith from this store is 96*2 bits/ 4 clocks (48 bits/clock). It is likely to be a 1 port memory; we use 1 clock to load the ALU instruction, 1 clocks to load the Fetch instruction, 1 clock to load 2 control flow instructions and 1 clock to write instructions.

4. Sequencer Instructions

All control flow instructions and move instructions are handled by the sequencer only. The ALUs will perform NOPs during this time (MOV PV,PV, PS,PS).

5. Constant Store

The constant store is managed by the CP. The sequencer is aware of where the constants are using a remaping table also managed by the CP. A likely size for the constant store is 512x128 bits. The constant store is also planned to be shared. The read BW from the constant store is 128 bits/clock and the write bandwith is 32/4 bits/clock.

In order to do constant store indexing, the sequencer must be loaded first with the indexes (that come from the GPRs). There are 144 wires from the exit of the SP to the sequencer (9 bits pointers x 16 vertexes/clock). Since the data must pass thru the Shader pipe for the float to fixed convertion, there is a latency of 4 clocks (1 instruction) between the time the sequencer is loaded and the time one can index into the constant store. The assembly will look like this

```
MOVA  R1.X,R2.X        // Loads the sequencer with the content of R2.X, also copies the content of R2.X into R1.X
NOP                    // latency of the float to fixed conversion
ADD   R3,R4,C0[R2.X]// Uses the state from the sequencer to add R4 to C0[R2.X] into R3
```

Note that we don't really care about what is in the brackets because we use the state from the MOVA instruction. R2.X is just written again for the sake of simplicity.

The storage needed in the sequencer in order to support this feature is 2*64*9 bits = 1152 bits.

6. Looping and Branches

Loops and branches are planned to be supported and will have to be dealt with at the sequencer level. We plan on supporting constant loops and branches using a control program.

6.1 The controlling state.

As per Dx9 the following state is available for control flow:

Boolean[15:0]
loop_count[7:0][7:0]
 In addition:
loop_start [7:0] [7:0]
loop_step [7:0] [7:0]
 Exist to give more control to the controlling program.

We will extend that in the R400 to:

Boolean[255:0]
Loop_count[7:0][15:0]
Loop_Start[7:0] [15:0] times 2 (one for constant,registert)
Loop_Step[7:0] [15:0] times 2 (one for constant,register)
Loop_End[7:0] [15:0]

{ISSUE: How is the controlling state loaded and how many contexts do we have?}

We have a stack of 4 elements for calling subroutines and 4 loop counters to allow for nested loops.

~~We also keep 8 predicate vectors and 8 AND/OR sets of 3 bits. These bits can be 0: all 0s, 1: all ones and 11: mixed.~~

6.2 The Control Flow Program

The R300 uses a match method for control flow: The shader is executed, and at every instruction its address is compared with addresses (or address?) in a control table. The “event” in the control table can redirect operations in the program.

The Method chosen for the R400 is a “control program”. The control program has ten basic instructions:

Execute
Conditional_execute
Conditional_Execute_Predicates
~~Conditional_execute_or_Jump~~
Conditional_jump
Call
Return
Loop_start
Loop_end
End_of_clause
NOP

Execute, causes the specified number of instructions in instruction store to be executed.
Conditional_execute checks a condition first, and if true, causes the specified number of instructions in instruction store to be executed.
Loop_start resets the corresponding loop counter to the start value on the first pass after it checks for the end condition and if met jumps over to a specified address.
Loop_end increments (decrements?) the loop counter and jumps back the specified number of instructions.
Call jumps to an address and pushes the IP counter on the stack. On the return instruction, the IP is poped from the stack.
Conditional_execute_or_Jump executes a block of instructions or jumps to an address is the condition is not met.
Conditional_execute_Predicates executes a block of instructions if all bits in the predicate vectors meet the condition.
End_of_clause marks the end of a clause.
Conditional_jumps jumps to an address if the condition is met.
NOP is a regular NOP

NOTE THAT ALL JUMPS MUST JUMP TO EVEN CFP ADDRESSES. Thus the compiler must insert NOPs where needed to align the jumps on even CFP addresses.

Also if the jump is logically bigger than pshader_cntl_size (or vshader_cntl_size) we break the program (clause) and set the debug registers. If an execute or conditional_execute is lower than cntl_size or bigger than size we also break the program (clause) and set the debug registers.

We have to fit instructions into 48 bits in order to be able to put two control flow instruction per line in the instruction store.

Execute				
47	46… 42	41 … 24	23 … 12	11 … 0

Addressing	00001	RESERVED	Instruction _count	Exec Address

Execute up to 4k instructions at the specified address in the instruction memory.

NOP		
47	46 … 42	41 … 0
Addressing	00010	RESERVED

If the specified boolean (8 bits can address 256 booleans) meets the specified condition then execute the specified instructions (up to 512 instructions) or if the condition is not met jump to the jump address in the control flow program. This MUST be a forward jump.

Conditionnal_Execute						
47	46 … 42	41 … 34	33	32 … 24	23 … 12	11 … 0
Addressing	00011	Boolean address	Condition	RESERVED	Instruction_count	Exec Address

If the specified boolean (8 bits can address 256 booleans) meets the specified condition then execute the specified instructions (up to 4k instructions)

Conditionnal_Execute_Predicates						
47	46 … 42	41 … 38	37	36 … 24	23 … 12	11 … 0
Addressing	00100	Predicate vector	Condition	RESERVED	Instruction_count	Exec Address

Check the AND/OR of all current predicate bits. If AND/OR matches the condition execute the specified number of instructions.

Loop_Start				
47	46 … 42	41 … 16	15 … 4	3 … 0
Addressing	00101	RESERVED	Jump address	Loop ID

Loop Start. Compares the loop count with the end value. If loop condition not met jump to the address. Forward jump only. Also computes the index value.

Loop_End				
47	46 … 42	41 … 16	15 … 4	3 … 0
Addressing	00111	RESERVED	Start address	Loop ID

Loop end. Increments the counter by one and jumps BACK only to the start of the loop.

The way this is described does not prevent nested loops, and the inclusion of the loop id make this easy to do.

Call			
47	46 … 42	41…12	11 … 0
Addressing	01000	RESERVED	Address

Jumps to the specified address and pushes the IP counter on the stack.

Return		
47	46 … 42	41 … 0
Addressing	01001	RESERVED

Pops the topmost address from the stack and jumps to that address. If nothing is on the stack, the program will just continue to the next instruction.

Conditionnal_Jump						
47	46 … 42	41 … 34	33	32 … 13	12	11 … 0
Addressing	01010	Boolean address	Condition	RESERVED	FW only	Address

If condition met, jumps to the address. FORWARD jump only allowed if bit 12 set. Bit 12 is only an optimization for the compiler and should NOT be exposed to the API.

End_of_Clause		
47	46 … 42	41 … 0

Addressing	01011	RESERVED

Marks the end of a clause.

To prevent infinite loops, we will keep 9 bits loop counters instead of 8 (we are only able to loop 256 times). If the counter goes higher than 255 then the loop_end or the loop_start is going to break the loop and set de debug registers. The sequencer will keep two loop indexes values:

IC index for constant indexing (9 bits)
IR index for register file indexing (7 bits)

This will be updated ~~everytime~~every time we loop and can only be used to index the constant store and the register file. The way to compute this value is:

Index = Loop_counter*Loop_iterator + Loop_init.

The IC for constant is going to return 0 if it is out of the constant range. The IR index is going to break the program if the index ~~exeeds~~exceeds the number of requested registers.

The basic model is as follows:

The render state defined the clause boundaries:

Vertex_shader_fetch[7:0][7:0]	// eight 8 bit pointers to the location where each clauses control program is located
Vertex_shader_alu[7:0][7:0]	// eight 8 bit pointers to the location where each clauses control program is located
Pixel_shader_fetch[7:0][7:0]	// eight 8 bit pointers to the location where each clauses control program is located
Pixel_shader_alu[7:0][7:0]	// eight 8 bit pointers to the location where each clauses control program is located

A pointer value of FF means that the clause doesn't contain any instructions.

The control program for a given clause is executed to completion before moving to another clause, (with the exception of the pick two nature of the alu execution). The control program is the only program aware of the clause boundaries.

6.3 Data dependant predicate instructions

Data dependant conditionals will be supported in the R400. The only way we plan to support those is by supporting three vector/scalar predicate operations of the form:

PRED_SETE_# - similar to SETE except that the result is 'exported' to the sequencer.
PRED_SETGT_# - similar to SETGT except that the result is 'exported' to the sequencer
PRED_SETGTE_# - similar to SETGTE except that the result is 'exported' to the sequencer

For the scalar operations only we will also support the two following instructions:
PRED_SETE0_# – SETE0
PRED_SETE1_# – SETE1

The export is a single bit - 1 or 0 that is sent using the same data path as the MOVA instruction. The sequencer will maintain 4 sets of 64 bit predicate vectors (in fact 8 sets because we interleave two programs but only 4 will be exposed) and use it to control the write masking. This predicate is not maintained across clause boundaries. The # sign is used to specify ~~wich~~which predicate set you want to use 0 thru 3.

Then we have two conditional execute bits. The first bit is a conditional execute "on" bit and the second bit tells us if we execute on 1 or 0. For ~~exemple~~example, the instruction:

P0_ADD_# R0,R1,R2

Is only going to write the result of the ADD into those GPRs whose predicate bit is 0. Alternatively, P1_ADD_# would only write the results to the GPRs whose predicate bit is set. The use of the P0 or P1 without precharging the sequencer with a PRED instruction is undefined.

{Issue: do we have to have a NOP between PRED and the first instruction that uses a predicate?}

6.4 HW Detection of PV,PS

Formatted: Bullets and Numbering

Because of the control program, the compiler cannot detect statically dependant instructions. In the case of non-masked writes and subsequent reads the sequencer will insert uses of PV,PS as needed. This will be done by comparing the read address and the write address of consecutive instructions. For masked writes, the sequencer will insert NOPs wherever there is a dependant read/write.

The sequencer will also have to insert NOPs between PRED_SET and MOVA instructions and their uses.

~~6.4~~6.5 Register file indexing

Formatted: Bullets and Numbering

Because we can have loops in fetch clause, we need to be able to index into the register file in order to retrieve the data created in a fetch clause loop and use it into an ALU clause. The instruction will include the base address for register indexing and the instruction will contain these ~~controls :~~controls:

Bit7	Bit 6	
0	0	'absolute register'
0	1	'relative register'
1	0	'previous vector'
1	1	'previous scalar'

In the case of an absolute register we just take the address as is. In the case of a relative register read we take the base address and we add to it the loop_index and this becomes our new address that we give to the shader pipe.

6.6 Predicated Instruction support for Texture clauses

Formatted: Bullets and Numbering

For texture clauses, we support the following optimization: we keep 1 bit (thus 4 bits for the four predicate vectors) per predicate vector in the reservation stations. A value of 1 means that one ore more elements in the vector have a value of one (thus we have to do the texture fetches for the whole vector. A value of 0 means that no elements in the vector have his predicate bit set and we can thus skip over the texture fetch.

6.7 Debugging the Shaders

Formatted: Bullets and Numbering

In order to be able to debug the pixel/vertex shaders efficiently, we provide 3 methods.

6.7.1 *Method 1: Debugging registers*

Formatted: Bullets and Numbering

Current plans are to expose 2 debugging, or error notification, registers:
1. address register where the first error occurred
2. count of the number of errors

The sequencer will detect the following groups of errors:
- count overflow
- jump error
 relative jump address > size of the control flow program
 relative jump address > length of the shader program
- constant overflow
- register overflow
- call stack
 call with stack full
 return with stack empty

With two of the errors, a jump error or a register overflow will cause the program to break. In this case, a break means that a clause will halt execution, but allowing further clauses to be executed.

With the other errors, program can continue to run, potentially to worst-case limits.

If indexing outside of the constant range, causing an overflow error, the hardware is specified to return the value with an index of 0. This could be exploited to generate error tokens, by reserving and initializing the 0th register (or constant) for errors.

6.7.2 Method 2: Exporting the values in the GPRs (12)

Formatted: Bullets and Numbering

The sequencer will have a count register and an address register for this mode and 3 bits per clause specifying the execution mode for each clause. The modes can be :

1) Normal
2) Debug Kill
3) Debug Addr
4) Debug Count

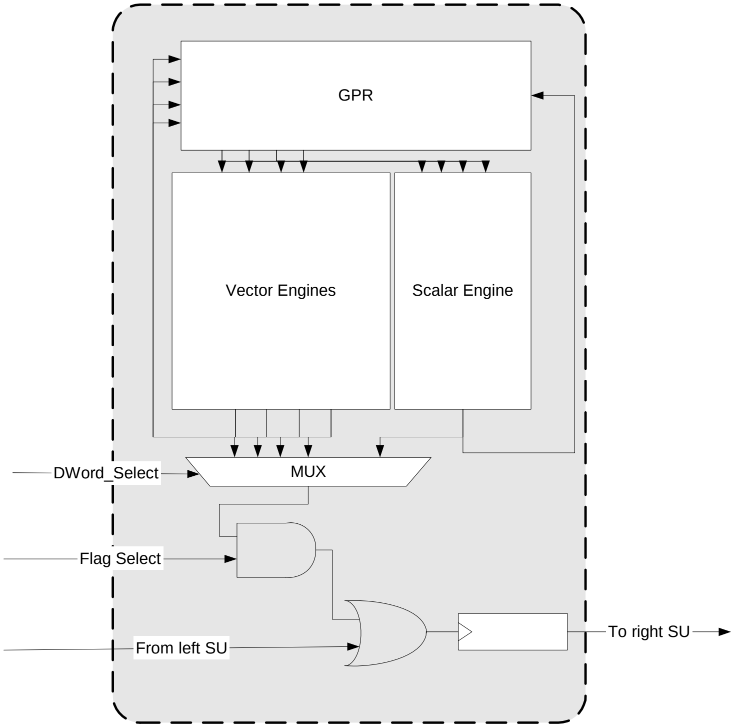
Formatted: Bullets and Numbering

Under the normal mode execution follows the normal course. Under the kill mode, all control flow instructions are executed but all normal shader instructions of the clause are replaced by NOPs. Only debug_export instructions of clause 7 will be executed under the debug kill setting. Under the two other modes, normal execution is done until we reach an address specified by the address register or an instruction count (useful for loops) specified by the count register. After we have hit the address or the count we change to the kill mode for the rest of the clause.

6.7.3 Method 3: Selective export of a 32 bit Dword.

Formatted: Bullets and Numbering

The third debug option will be mainly used for HW debug. For this mode, the sequencer will keep the following control debug registers: Shader_pipe (6 bits), Mode(1 bit), Dword_select (3 bits), clause_+count (16 bits?),address (12 bits) Vector_number (8 bits), Render_state (21 bits). The shader pipe register selects a shader pipe amongst the 64, the dword_select selects a channel (0…3 of vector or scalar), the clause_+count selects at which clause and which count we export, the Render_state specifies which render state is concerned, the Vector_number specifies which vector is concerned and the mode selects count export, or address export.



Flag Select is a combination of Shader pipe, clause +count, address, Vector number and render state. It is only active for 1 shader pipe at a time and for 1 vector of a given state. The driver is responsible to reset the output register to 0 before executing a given program.

7. Pixel Kill Mask

A vector of 64 bits is kept per group of pixels/vertices. Its purpose is to optimize the texture fetch requests and allow the shader pipe to kill pixels using the following instructions:

MASK_SETE
MASK_SETGT
MASK_SETGTE

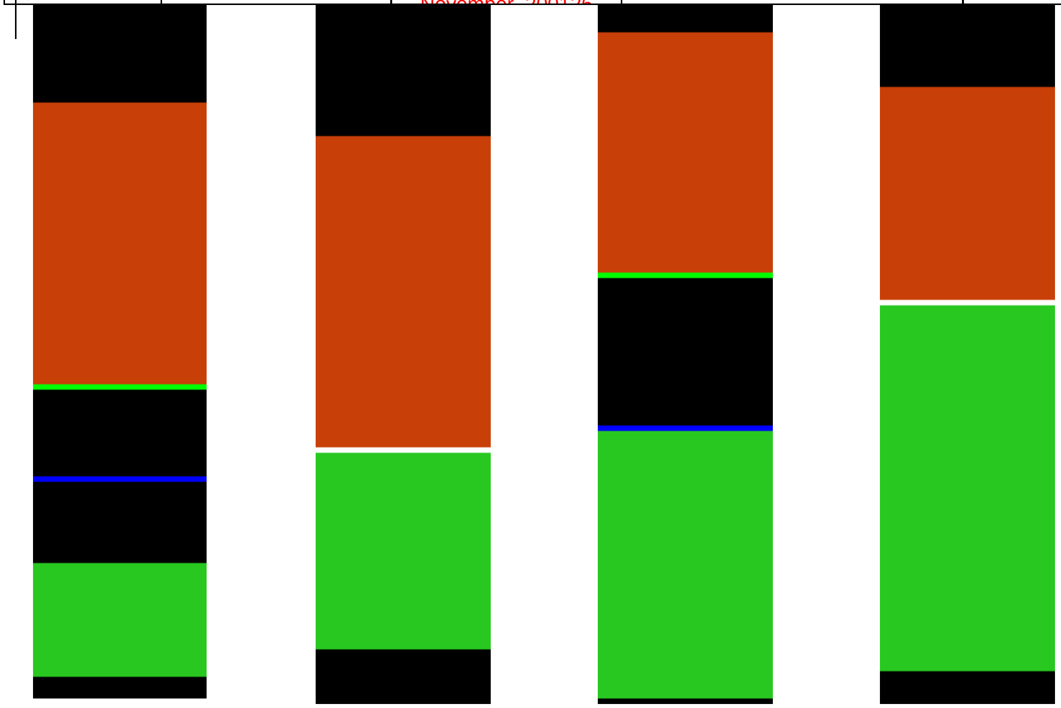
~~However, if the driver sets the kill_vector_on register to 0 (don't use) then the 64 bit kill mask becomes the 5th predicate vector and is kept across clause boundaries (thus allowing predicated instructions to be used in texture clauses). In this mode, the sequencer is going to send all 1s to the RBs for coverage mask information.~~

8. HOS surfaces

HOS surfaces are able to export from the 6 last clauses but to memory ONLY. If they want to export to the parameter cache they have to do it in the last clause (7). They can also export position in clause 3. ~~The buffer they want to export into must be specified in the "exports" field of the state registers.~~

9. Register file allocation

The register file allocation for vertices and pixels can either be static or dynamic. In both cases, the register file in managed using two round robins (one for pixels and one for vertices). In the dynamic case the boundary between pixels and vertices is allowed to move, in the static case it is fixed to VERTEX_REG_SIZE for vertices and 256-VERTEX_REG_SIZE for pixels.



Above is an example of how the algorithm works. Vertices come in from top to bottom; pixels come in from bottom to top. Vertices are in orange and pixels in green. The blue line is the tail of the vertices and the green line is the tail of the pixels. Thus anything between the two lines is shared. When pixels meets vertices the line turns white and the boundary is static until both vertices and pixels share the same "unallocated bubble". Then the boundary is allowed to move again.

10. Fetch Arbitration

The fetch arbitration logic chooses one of the 8 potentially pending fetch clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute. Once chosen, the clause state machine will send one 2x2 fetch per clock (or 4 fetches in one clock every 4 clocks) until all the fetch instructions of the clause are sent. This means that there cannot be any dependencies between two fetches of the same clause.

The arbitrator will not wait for the fetches to return prior to selecting another clause for execution. The fetch pipe will be able to handle up to X(?) in flight fetches and thus there can be a fair number of active clauses waiting for their fetch return data.

11. ALU Arbitration

ALU arbitration proceeds in almost the same way than fetch arbitration. The ALU arbitration logic chooses one of the 8 potentially pending ALU clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute. There are two ALU arbitrers, one for the even clocks and one for the odd clocks. For exemple, here is the sequencing of two interleaved ALU clauses (E and O stands for Even and Odd sets of 4 clocks):

Einst0 Oinst0 Einst1 Oinst1 Einst2 Oinst2 Einst0 Oinst3 Einst1 Oinst4 Einst2 Oinst0…
Proceeding this way hides the latency of 8 clocks of the ALUs.

12. Handling Stalls

When the output file is full, the sequencer prevents the ALU arbitration logic to select the last clause (this way nothing can exit the shader pipe until there is place in the output file. If the packet is a vertex packet and the position buffer is full (POS_FULL) then the sequencer also prevents a thread to enter the exporting clause (4?). The sequencer will set the OUT_FILE_FULL signal n clocks before the output file is actually full and thus the ALU arbitrer will be able read this signal and act accordingly by not preventing exporting clauses to proceed.

13. Content of the reservation station FIFOs

21 bits of Render State 7 bits for the base address of the GPRs, some bits for LOD correction and coverage mask information in order to fetch fetch for only valid pixels, quad address and 1 bit to specify if the vector is of pixels or vertices. Since pixels and vertices are kept in order in the shader pipe, we only need two fifos (one for vertices and one for pixels) deep enough to cover the shader pipe latency. This size will be determined later when we will know the size of the small fifos between the reservation stations.

14. The Output File

The output file is where pixels are put before they go to the RBs. The write BW to this store is 256 bits/clock. Just before this output file are staging registers with write BW 512 bits/clock and read BW 256 bits/clock. For this reason only ONE concurrent program can be of clause 8 (exporting clause) the other program MUST not. The staging registers are 4x128 (and there are 16 of those on the whole chip).

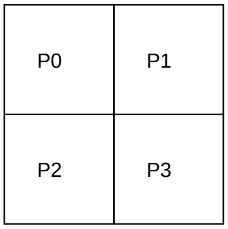
15. IJ Format

The IJ information sent by the PA is of this format on a per quad basis:

We have a vector of IJ's (one IJ per pixel at the centroid of the fragment or at the center of the pixel depending on the mode bit). The interpolation is done at a different precision across the 2x2. The upper left pixel's parameters are always interpolated at full ~~19x24~~ 20x24 mantissa precision. Then the result of the interpolation along with the difference in IJ in reduced precision is used to interpolate the parameter for the other three pixels of the 2x2. Here is how we do it:

Assuming P0 is the interpolated parameter at Pixel 0 having the barycentric coordinates I(0), J(0) and so on for P1,P2 and P3. Also assuming that A is the parameter value at V0 (interpolated with I), B is the parameter value at V1 (interpolated with J) and C is the parameter value at V2 (interpolated with (1-I-J).

$$\Delta 01I = I(1) - I(0)$$
$$\Delta 01J = J(1) - J(0)$$
$$\Delta 02I = I(2) - I(0)$$
$$\Delta 02J = J(2) - J(0)$$
$$\Delta 03I = I(3) - I(0)$$
$$\Delta 03J = J(3) - J(0)$$



$$P0 = C + I(0)*(A - C) + J(0)*(B - C)$$
$$P1 = P0 + \Delta 01I*(A - C) + \Delta 01J*(B - C)$$
$$P2 = P0 + \Delta 02I*(A - C) + \Delta 02J*(B - C)$$
$$P3 = P0 + \Delta 03I*(A - C) + \Delta 03J*(B - C)$$

P0 is computed at ~~19x24~~ 20x24 mantissa precision and P1 to P3 are computed at 8X24 mantissa precision. So far no visual degradation of the image was seen using this scheme.

Multiplies (Full Precision): 2
Multiplies (Reduced precision): 6
Subtracts 19x24 (Parameters): 2
Adds: 8

FORMAT OF P0’s IJ : Mantissa ~~19~~ 20 Exp 4 for I + Sign
Mantissa ~~19~~ 20 Exp 4 for J + Sign

FORMAT of Deltas (x3):Mantissa 8 Exp 4 for I + Sign
Mantissa 8 Exp 4 for J + Sign

Total number of bits : 19*2 + 8*6 + 4*8 + 4*2 = 128

The Deltas have a leading 1, the Full precision IJs don’t. This means that in the case of the deltas we MUST be able to shift 8 right (exponent value of 0 means number = 0, exponent value of 1 means shift right 8). This means that the maximum range for the IJs (Full precision) is +/- 64 and the range for the Deltas is +/- 128.~~6~~

16. The parameter cache

The parameter cache is where the vertex shaders export their data. It consists of 16 128x128 memories (1R/1W). The reuse engine will make it so that all vertexes of a given primitive will hit different memories.

17. Vertex position exporting

On clause 4 (or 5) the vertex shader can export to the PA both the vertex position and the point sprite. It can also do so at clause 8 if not done at clause 4. Along with the position is exported a pointer to the parameter cache where the data will be once the vertex shader exports. The storage needed to perform the position export is at least 64x128 memories for the position and 64x32 memories for the sprite size. It is going to be taken in the pixel output fifo.

18. Exporting Arbitration

Here are the rules for co-issuing exporting ALU clauses.

1) Position exports and position exports cannot be co-issued.
2) Position exports and memory exports cannot be co-issued.
3) Position exports and Z/Color exports cannot be co-issued.
4) Memory exports and Z/Color exports cannot be co-issued.
5) Memory exports and memory exports cannot be co-issued.
6) Z/color exports and Z/color exports cannot be co-issued.
7) Parameter exports and Z/Color exports CAN be co-issued.
8) Parameter exports and parameter exports CAN be co-issued.
9) Parameter exports and memory exports CAN be co-issued.

19. Real time commands

We are unable to use the parameter memory since there is no way for a command stream to write into it. Instead we need to add three 16x128 memories (one for each of three vertices x 16 interpolants). These will be mapped onto the register bus and written by type 0 packets, and output to the the parameter busses (the sequencer and/or PA need to be able to address the realtime parameter memory as well as the regular parameter store. For higher performance we should be able able to view them as two banks of 16 and do double buffering allowing one to be loaded, while the other is rasterized with. Most overlay shaders will need 2 or 4 scalar coordinates, one option might be to restrict the memory to 16x64 or 32x64 allowing only two interpolated scalars per cycle, the only problem I see with this is, if we view support for 16 vector-4 interpolants important (true only if we map microsoft's high priority stream to the realtime stream), then the PA/sequencer need to support a realtime-specific mode where we need to address 32 vectors of parameters instead of 16.

20. Registers

20.1 Control

DYNAMIC_REG	Dynamic allocation (pixel/vertex) of the register file on or off.
VERTEX_REG_SIZE	What portion of the register file is reserved for vertices (static allocation only)
PIXEL_MIN_SIZE	Minimal size of the register file's pixel portion (dynamic only)
VERTEX_MIN_SIZE	Minimal size of the register file's vertex portion (dynamic only)
ARBITRATION_policy	policy of the arbitration between ~~vetexes~~vertexes and pixels
CST_SIZE_P	Size of the constant store for pixels
CST_SIZE_V	Size of the constant store for vertexes
INST_STOR_ALLOC	interleaved, separate, interleaved+shared,separate+shared
~~VWRAP~~PWRAP	~~wrap~~start point for the ~~vertex~~pixel shader instruction store (vertex shader always starts at 0)
~~PWRAP~~SHAREDWRAP	~~wrap~~start point for the ~~pixel shader~~shared instruction store
RTWRAP	start point for the RT instruction store (RT always ends at the end of the IM)
NO_INTERLEAVE	debug state register. Only allows one program at a time into the GPRs

20.2 Context

Vshader_fetch[7:0][7:0]	eight 8 bit pointers to the location where each clauses control program is located
Vshader_alu[7:0][7:0]	eight 8 bit pointers to the location where each clauses control program is located
Pshader_fetch[7:0][7:0]	eight 8 bit pointers to the location where each clauses control program is located
Pshader_alu[7:0][7:0]	eight 8 bit pointers to the location where each clauses control program is located
PSHADER	base pointer for the pixel shader
VSHADER	base pointer for the vertex shader
Vshader_cntl_size	size of the vertex shader (# of instructions in control program/2)
Pshader_cntl_size	size of the pixel shader (# of instructions in control program/2)
Pshader_size	size of the pixel shader (cntl+instructions)
Vshader_size	size of the vertex shader (cntl+instructions)
REG_ALLOC_PIX	number of registers to allocate for pixel shader programs
REG_ALLOC_VERT	number of registers to allocate for vertex shader programs
FLAT_GOUR[0…15]	~~wich~~which parameters are to be gouraud shaded
~~GEN_TEX~~	~~Do we generate texture coordinates for 1st parameter or not~~
CYL_WRAP[0…63]	for ~~wich~~which parameters (and channels (xyzw)) do we do the cyl wrapping.
P_export_mode	0xxxx : Normal mode 1xxxx : Multipass mode If normal, bbbz where bbb is how many colors (0-4) and z is export z or not If multipass 1-12 exports for color.
vshader_export_mask	~~wich~~which of the last 6 ALU clauses is exporting
vshader_export_mode	0: position (1 vector), 1: position (2 vectors), 3:multipass
vshader_export_count[6]	# of interpolated parameters exported in clause 7 OR # of exported vectors to memory per clause in multipass mode (per clause)
kill_vector_on	use the mask kill vector to kill pixels and optimize texture pipe fetches OR use it as the fifth predicate vector ~~wich~~which is the only predicate vector kept across clause boundaries.

21. DEBUG registers

21.1 Control

Formatted: Bullets and Numbering

Shader_pipe	# of the shader pipe for method 3 (0…64)
Count_+clause	instruction count and clause number for method 3
Dword_select	channel select for method 3

Mode	operating mode for method 3
Rstate	render state method 3 is operating on
Vector_count	vector number the method 3 will export

21.2 Context

Formatted: Bullets and Numbering

PROB_ADDR	instruction address where the first problem occurred
PROB_COUNT	number of problems encountered during the execution of the program
Count	instruction counter for debug method 2
Clause_mode[3]	clause mode for debug method 2

22. Interfaces

22.1 External Interfaces

22.1.1 *PA/SC to SP0 : IJ bus*

This is a bus that sends the IJ information to the IJ fifos on the top of each shader pipe. At the same time the control information goes to the sequencer. There are 4 of these buses over the whole chip (SP0 thru 3)

Name	Direction	Bits	Description
IJs	PA→SP0	64~~3~~	IJ information sent over 2 clocks
Mask	PA→SP0	1	Write Mask

22.1.2 *PA/SC to SEQ : IJ Control bus*

This is the control information sent to the sequencer in order to control the IJ fifos and all other information needed to execute a shader program on the sent pixels.

Name	Direction	Bits	Description
Write Mask	PA→SEQ(SP)	4	Quad Write mask left to right
LOD_CORRECT	PA→SEQ(SP)	24	LOD correction per quad (6 bits per quad)
FVTX	PA→SEQ(SP)	2	Provoking vertex for flat shading
PPTR0	PA→SEQ(SP)	11	P Store pointer for vertex 0
PPRT1	PA→SEQ(SP)	11	P Store pointer for vertex 1
PPTR2	PA→SEQ(SP)	11	P Store pointer for vertex 2
E_OFF_VECTOR	PA→SEQ(SP)	1	End of the vector
DEALLOC	PA→SEQ(SP)	1	Deallocation token for the P Store
STATE	PA→SEQ(SP)	21	State/constant pointer (6*3+3)
VALID	PA→SEQ(SP)	16	Valid bits for all pixels
NULL	PA→SEQ(SP)	1	Null Primitive (for PC deallocation purposes)
E_OFF_PRIM	PA→SEQ(SP)	1	End Of the primitive
FBFACE	PA→SEQ(SP)	1	Front face = 1, back face = 0
TYPE	PA→SEQ(SP)	3	Stippled line and Real time command need to load tex cords from alternate buffer 000 : Normal 001 : Stippled line 011 : Real Time 100 : Line AA 101 : Point AA 110 : Sprite
RTRn	SEQ→PA	1	Stalls the PA in n clocks
RTS	PA→SEQ(SP)	1	PA ready to send data

22.1.3 *SEQ to SP0 : Interpolator bus*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
TYPE	SEQ→SP0	3	Type of the primitive

Formatted

			000 : Normal 001 : Stippled line/Poly 011 : Real Time 100 : Line AA 101 : Point AA 110 : Sprite
FVTX	SEQ→SP0	2	Provoking vertex for flat shading
FLAT_GOURAUD	SEQ→SP0	1	Flat or gouraud shading
CYL_WRAP	SEQ→SP0	4	Wich parameter needs to be cylindrical wrapped
Inter_Write_Register_Index	SEQ→SP0	7	Index into Register file for write of Interpolated data
Write_Mask	SEQ→SP0	4	Quad Write mask left to right
IJ_Line number	SEQ→SP0	2	Line in the IJ buffer to use to interpolate
Swap_Buffers	SEQ→SP0	1	Swap the IJ buffers at the end of the interpolation
Param_0	SEQ→SP0	1	We are interpolating parameter 0

Formatted
Formatted
Formatted
Formatted
Formatted

22.1.4 *SEQ to SP0 : Parameter Cache bus*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
Ptr1	SEQ→SP0	7	Pointer of PC (7 LSBs of Pointer)
Ptr2	SEQ→SP0	7	Pointer of PC (7 LSBs of Pointer)
Ptr3	SEQ→SP0	7	Pointer of PC (7 LSBs of Pointer)

22.1.5 *SEQ to SX0 : Parameter Cache Mux control Bus*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
Mux1	SEQ→SX0	4	Mux control for PC (4 MSbs of Pointer)
Mux2	SEQ→SX0	4	Mux control for PC (4 MSbs of Pointer)
Mux3	SEQ→SX0	4	Mux control for PC (4 MSbs of Pointer)

Formatted
Formatted
Formatted
Formatted
Formatted

22.1.6 *SX0 to SP0 : Parameter Cache Return bus*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
Vtx_data_1	SX0→SP0	128	Vertex data to interpolate
Vtx_data_2	SX0→SP0	128	Vertex data to interpolate
Vtx_data_3	SX0→SP0	128	Vertex data to interpolate

~~22.1.3~~22.1.7 *VGT to SP0/SEQ : Vertex Bus*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
Vertex indexes	VGT→~~RE~~SP0	128	Pointers of indexes or HOS surface information
EOF_vector	VGT→~~RE~~SP0/SEQ	1	End of the vector
Inputs_vert	VGT→~~RE~~SP0/SEQ	1	0: Normal 128 bits per vert 1: double 256 bits per vert
STATE	VGT→SEQ	21	Render State (6*3+3 for constants)

~~22.1.4~~22.1.8 *CP to SEQ : Constant store load*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
Constant Address	CP→SEQ	8	Address of the block of 4 constants
Constant Data	CP→SEQ	512	Data sent over 4 clocks
Remap Address	CP→SEQ	10	Remaping address write address
Remap Data pointer	CP→SEQ	8	Remaping pointer

~~22.1.5~~22.1.9 *CP to SEQ : Fetch State store load*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
Constant Address	CP→SEQ	8	Address of the block of 4 state constants
Constant Data	CP→SEQ	512	Data sent over 4 clocks
Remap Address	CP→SEQ	10	Remaping address write address

Remap Data pointer	CP→SEQ	8	Remaping pointer

~~22.1.6~~22.1.10 CP to SEQ : Control State store load

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
Constant Address	CP→SEQ	?	
Constant Data	CP→SEQ	?	

Formatted

Formatted

Formatted

{ISSUE: How,Who and what is the size of this bus?}

~~22.1.7~~22.1.11 MH to SEQ: Instruction store Load

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
Instruction address	MH→SEQ	12	Instruction address
Instruction	MH→SEQ	96	Instruction X times
Control Instruction address	MH→SEQ	9	Pointer to the control instruction store
Control Instruction	MH→SEQ	32	Control Instruction X times

{ISSUE: CP or MH?}

~~22.1.8~~22.1.12 SP0 to ~~RB~~ SX0 : Pixel read from RBs

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
Export_data	SP0→~~RB~~SX0	64*16	32 pairs of 32 bits channel values
Shader_Dest	SP0→SX0~~SP→RB~~	4	Specifies one of the of up to 12 export destinations
Shader_Count	SP0→SX0~~SP→RB~~	3	Each set of four pixels or vectors is exported over eight clocks. This field specifies where the SP is in that sequence.
Shader_Last	SP0→SX0~~SP→RB~~	1	The current export clause is over (true for one clock) The last export instruction creates *two* cycles to the RB. This needs to be set on or after the last RB cycle that is produced by the last export instruction, but before the first RB cycle of the first export instruction of the next clause.
Shader_PixelValid	SP0→SX0~~SP→RB~~	4x4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
Shader_WordValid	SP0→SX0~~SP→RB~~	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors

~~22.1.9~~22.1.13 SEQ to ~~RB~~ SX0 : Control bus

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
Export_Pixel	SEQ→~~RB~~SX0	1	1: Pixel 0: Vertex
Export_SEND	SEQ→SX0~~SEQ→RB~~	1	Raised to indicate that the SQ is starting an export
Export_Clause	SEQ→SX0~~SEQ→RB~~	3	Clause number, which is needed for vertex clauses
Export_State	SEQ→SX0~~SEQ→RB~~	21?	State ID, which is needed for vertex clauses

These fields are sent synchronously with SP export data, described in SP0→~~RB~~ SX0 interface

{ISSUE: Where are the PC pointers}

~~22.1.10~~22.1.14 ~~RB~~ SX0 to SEQ : Output file control

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
Export_RTS	~~RB~~SX0→SEQ	1	Raised by ~~RB~~ SX0 to indicate that the following two fields reflect the result of the most recent export
Export_Position	SX0→SEQ~~RB→SEQ~~	1	Specifies whether there is room for another position.
Export_Buffer	SX0→SEQ~~RB→SEQ~~	7	Specifies the space availble in the output buffers. 0: buffers are full

			1: 2K-bits available (32-bits for each of the 64 pixels in a clause) ... 64: 128K-bits available (16 128-bit entries for each of 64 pixels) 65-127: RESERVED

Formatted: Bullets and Numbering

~~22.1.11~~22.1.15 *SP0 to ~~RB~~ SX0 : Position return bus*

Name	Direction	Bits	Description
Position return	SP0→~~RB~~SX0	128	Position data or sprite size (per clock)

~~For point sprites and position exports the size and position are interleaved on a 16 x 16 basis. We export 1 position then 1 point sprite sizes. The storage used is of 64x128 bits for position and 64x32 bits for sprite size, it is taken from the output buffer. Additionnally,if needed the edge flags are packed into the bits of the sprite sizes.~~

Formatted: Bullets and Numbering

~~22.1.12~~22.1.16 *Shader Engine to Fetch Unit Bus (Fast Bus)*

Four quad's worth of addresses is transferred to Fetch Unit every clock. These are sourced from a different pixel within each of the sub-engines repeating every 4 clocks. The register file index to read must precede the data by 2 clocks. The Read address associated with Quad 0 must be sent 1 clock after the Instruction Start signal is sent, so that data is read 3 clocks after the Instruction Start.

Four Quad's worth of Fetch Data may be written to the Register file every clock. These are directed to a different pixel of the sub-engines repeating every 4 clocks. The register file index to write must accompany the data. Data and Index associated with the Quad 0 must be sent 3 clocks after the Instruction Start signal is sent.

Name	Direction	Bits	Description
Tex_RegFile_Read_Data	SP->TEX	2048	16 Fetch Addresses read from the Register file
Tex_RegFile_Write _Data	TEX→SP	2048	16 texture results

Formatted: Bullets and Numbering

~~22.1.13~~22.1.17 *Sequencer to Fetch Unit bus (Slow Bus)*

Once every four clock, the fetch unit sends to the sequencer on wich clause it is now working and if the data in the registers is ready or not. This way the sequencer can update the fetch counters for the reservation station fifos. The sequencer also provides the intruction and constants for the fetch to execute and the address in the register file where to write the fetch return data.

Name	Direction	Bits	Description
Tex_Ready	TEX→ SEQ	1	Data ready
Tex_Clause_Num	TEX→ SEQ	3	Clause number
Tex_cst	SEQ→TEX	10	Fetch state address 10 bits sent over 4 clocks
Tex_Inst	SEQ→TEX	12	Fetch instruction address 12 bits sent over 4 clocks
EO_CLAUSE	SEQ→TEX	1	Last instruction of the clause
PHASE	SEQ→TEX	1	Write phase signal
LOD CORRECT	SEQ→TEX	96	LOD correct 3 bits per comp 2 components per quad * 16 quads
Mask	SEQ→TEX	64	Pixel mask 1 bit per pixel
Tex_Clause_Num	SEQ→TEX	3	Clause number
Tex_Write_Register_Index	SEQ->TEX	7	Index into Register file for write of returned Fetch Data
Tex_Read_Register_Index	SEQ->SP	7	Index into Register files for reading Fetch Address (internal)

Formatted: Bullets and Numbering

23. ~~Internal interfaces~~

23.1.1 ~~*RE to SEQ : Vertex Control Bus*~~

~~This information is sent over 4 clocks.~~

Formatted: Bullets and Numbering

~~24.~~23. Examples of program executions

~~24.1.1~~23.1.1 *Sequencer Control of a Vector of Vertices*

1. PA sends a vector of 64 vertices (actually vertex indices – 32 bits/index for 2048 bit total) to the RE's Vertex FIFO
   - state pointer as well as tag into position cache is sent along with vertices
   - space was allocated in the position cache for transformed position before the vector was sent
   - **also before the vector is sent to the RE, the CP has loaded the global instruction store with the vertex shader program (using the MH?)**
   - The vertex program is assumed to be loaded when we receive the vertex vector.
     - the SEQ then accesses the IS base for this shader using the local state pointer (provided to all sequencers by the RBBM when the CP is done loading the program)
2. SEQ arbitrates between the Pixel FIFO and the Vertex FIFO – basically the Vertex FIFO always has priority
   - at this point the vector is removed from the Vertex FIFO
   - the arbitrer is not going to select a vector to be transformed if the parameter cache is full unless the pipe as nothing else to do (ie no pixels are in the pixel fifo).
3. SEQ allocates space in the SP register file for index data plus GPRs used by the program
   - the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer that came down with the vertices
   - SEQ will not send vertex data until space in the register file has been allocated
4. SEQ sends the vector to the SP register file over the RE_SP interface (which has a bandwidth of 2048 bits/cycle)
   - the 64 vertex indices are sent to the 64 register files over 4 cycles
     - RF0 of SU0, SU1, SU2, and SU3 is written the first cycle
     - RF1 of SU0, SU1, SU2, and SU3 is written the second cycle
     - RF2 of SU0, SU1, SU2, and SU3 is written the third cycle
     - RF3 of SU0, SU1, SU2, and SU3 is written the fourth cycle
   - the index is written to the least significant 32 bits **(floating point format?) (what about compound indices)** of the 128-bit location within the register file (w); the remaining data bits are set to zero (x, y, z)
5. SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of fetch state machine 0, or TSM0 FIFO)
   - the control packet contains the state pointer, the tag to the position cache and a register file base pointer.
6. TSM0 accepts the control packet and fetches the instructions for fetch clause 0 from the global instruction store
   - TSM0 was first selected by the TSM arbiter before it could start
7. all instructions of fetch clause 0 are issued by TSM0
8. the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)
   - TSM0 does not wait for requests made to the Fetch Unit to complete; it passes the register file write index for the fetch data to the TU, which will write the data to the RF as it is received
   - once the TU has written all the data to the register files, it increments a counter that is associated with ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go ahead start to execute the ALU clause
9. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store
10. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of fetch state machine 1, or TSM1 FIFO)
11. the control packet continues to travel down the path of reservation stations until all clauses have been executed
    - position can be exported in ALU clause 3 (or 4?); the data (and the tag) is sent over a position bus (which is shared with all four shader pipes) back to the PA's position cache
    - A parameter cache pointer is also sent along with the position data. This tells to the PA where the data is going to be in the parameter cache.
      - there is a position export FIFO in the SP that buffers position data before it gets sent back to the PA

    - the ASM arbiter will prevent a packet from starting an exporting clause if the position export FIFO is full
- parameter data is exported in clause 7 (as well as position data if it was not exported earlier)
    - parameter data is sent to the Parameter Cache over a dedicated bus
    - the SEQ allocates storage in the Parameter Cache, and the SEQ deallocates that space when there is no longer a need for the parameters (it is told by the PA when using a token).
    - the ASM arbiter will prevent a packet from starting on ASM7 if the parameter cache (or the position buffer if position is being exported) is full

12. after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program

~~24.1.2~~23.1.2 *Sequencer Control of a Vector of Pixels*

Formatted: Bullets and Numbering

1. **As with vertex shader programs, pixel shaders are loaded into the global instruction store by the CP**
    - At this point it is assumed that the pixel program is loaded into the instruction store and thus ready to be read.
2. the RE's Pixel FIFO is loaded with the barycentric coordinates for pixel quads by the detailed walker
    - the state pointer and the LOD correction bits are also placed in the Pixel FIF0
    - the Pixel FIFO is wide enough to source four quad's worth of barycentrics per cycle
3. SEQ arbitrates between Pixel FIFO and Vertex FIFO – when there are no vertices pending OR there is no space left in the register files for vertices, the Pixel FIFO is selected
4. SEQ allocates space in the SP register file for all the GPRs used by the program
    - the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer
    - SEQ will not allow interpolated data to be sent to the shader until space in the register file has been allocated
5. SEQ controls the transfer of interpolated data to the SP register file over the RE_SP interface (which has a bandwidth of 2048 bits/cycle). See interpolated data bus diagrams for details.
6. SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of fetch state machine 0, or TSM0 FIFO)
    - note that there is a separate set of reservation stations/arbiters/state machines for vertices and for pixels
    - the control packet contains the state pointer, the register file base pointer, and the LOD correction bits
    - all other informations (such as quad address for example) travels in a separate FIFO
7. TSM0 accepts the control packet and fetches the instructions for fetch clause 0 from the global instruction store
    - TSM0 was first selected by the TSM arbiter before it could start
8. all instructions of fetch clause 0 are issued by TSM0
9. the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)
    - TSM0 does not wait for fetch requests made to the Fetch Unit to complete; it passes the register file write index for the fetch data to the TU, which will write the data to the RF as it is received
    - once the TU has written all the data for a particular clause to the register files, it increments a counter that is associated with the ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go ahead and pop the FIFO and start to execute the ALU clause
10. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store
11. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of fetch state machine 1, or TSM1 FIFO)
12. the control packet continues to travel down the path of reservation stations until all clauses have been executed
    - pixel data is exported in the last ALU clause (clause 7)
        - it is sent to an output FIFO where it will be picked up by the render backend
        - the ASM arbiter will prevent a packet from starting on ASM7 if the output FIFO is full
13. after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program

~~24.1.3~~23.1.3 *Notes*

Formatted: Bullets and Numbering

14. The state machines and arbitrers will operate ahead of time so that they will be able to immediately start the real threads or stall.
15. The register file base pointer for a vector needs to travel with the vector through the reservation stations, but the instruction store base pointer does not – this is because the RF pointer is different for all threads, but the IS pointer is only different for each state and thus can be accessed via the state pointer
16. Waterfalling still needs to be specked out.

~~25.~~24. Open issues

Formatted: Bullets and Numbering

There is currently an issue with constants. If the constants are not the same for the whole vector of vertices, we don't have the bandwith from the fetch store to feed the ALUs. Two solutions exists for this problem:

1) Let the compiler handle the case and put those instructions in a fetch clause so we can use the bandwith there to operate. This requires a significant amount of temporary storage in the register store.
2) Waterfall down the pipe allowing only at a given time the vertices having the same constants to operate in parrallel. This might in the worst case slow us down by a factor of 16.

Need to do some testing on the size of the register file as well as on the register file allocation method (dynamic VS static).

Saving power?

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 2015~~16~~ November, 20017	GEN-CXXXXX-REVA	1 of 35

Author: Laurent Lefebvre

Issue To:	**Copy No:**

R400 Sequencer Specification

SEQ

Version 1.2~~1~~

Overview: This is an architectural specification for the R400 Sequencer block (SEQ). It provides an overview of the required capabilities and expected uses of the block. It also describes the block interfaces, internal sub-blocks, and provides internal state diagrams.

AUTOMATICALLY UPDATED FIELDS:
Document Location: C:\perforce\r400\arch\doc\gfx\RE\R400_Sequencer.doc
Current Intranet Search Title: R400 Sequencer Specification

APPROVALS	
Name/Dept	Signature/Date

Remarks:

THIS DOCUMENT CONTAINS CONFIDENTIAL INFORMATION THAT COULD BE SUBSTANTIALLY DETRIMENTAL TO THE INTEREST OF ATI TECHNOLOGIES INC. THROUGH UNAUTHORIZED USE OR DISCLOSURE.

Table Of Contents

1. OVERVIEW 4
1.1 Top Level Block Diagram.................................6
1.2 Data Flow graph...8
1.3 Control Graph 9
2. INTERPOLATED DATA BUS 9
3. INSTRUCTION STORE.................................12
4. SEQUENCER INSTRUCTIONS.................14~~12~~
5. CONSTANT STORE.....................................14~~12~~
6. LOOPING AND BRANCHES.....................14~~12~~
6.1 The controlling state......................................14~~12~~
6.2 The Control Flow Program..........................15~~13~~
6.3 Data dependant predicate instructions ....17~~15~~
6.4 HW Detection of PV,PS ....18~~1516~~
6.5 Register file indexing...................................18~~16~~
6.6 Predicated Instruction support for Texture clauses..18~~16~~
6.7 Debugging the Shaders ....18~~16~~
6.7.1 Method 1: Debugging registers........18~~16~~
6.7.2 Method 2: Exporting the values in the GPRs (12) 19~~1717~~
6.7.3 Method 3: Selective export of a 32 bit Dword. 19~~1717~~
7. PIXEL KILL MASK ....20~~1718~~
8. HOS SURFACES ....20~~1718~~
9. REGISTER FILE ALLOCATION ....20~~1718~~
10. FETCH ARBITRATION ....21~~1819~~
11. ALU ARBITRATION..............................21~~1819~~
12. HANDLING STALLS.............................22~~1920~~
13. CONTENT OF THE RESERVATION STATION FIFOS22~~1920~~
14. THE OUTPUT FILE ....22~~1920~~
15. IJ FORMAT..22~~1920~~
16. THE PARAMETER CACHE ....23~~2021~~
17. VERTEX POSITION EXPORTING ....23~~2021~~
18. EXPORTING ARBITRATION................23~~2021~~
19. REAL TIME COMMANDS.....................23~~2021~~
20. REGISTERS ....24~~2122~~
20.1 Control..24~~2122~~
20.2 Context...24~~2122~~
21. DEBUG REGISTERS ....25~~2222~~
21.1 Control..25~~2222~~
21.2 Context ....25~~2222~~
22. INTERFACES ....25~~2223~~
22.1 External Interfaces ....25~~2223~~
22.1.1 PA/SC to SP0 : IJ bus ....25~~2223~~
22.1.2 PA/SC to SEQ : IJ Control bus.26~~2223~~
22.1.3 SEQ to SP0 : Interpolator bus ..26~~2323~~
22.1.4 SEQ to SP0 : Parameter Cache bus 27~~2324~~
22.1.5 SEQ to SX0 : Parameter Cache Mux control Bus 27~~2324~~
22.1.6 SX0 to SP0 : Parameter Cache Return bus 27~~2324~~
22.1.7 VGT to SP0/SEQ : Vertex Bus .28~~2324~~
22.1.8 CP to SEQ : Constant store load 28~~2424~~
22.1.9 CP to SEQ : Fetch State store load 28~~2424~~
22.1.10 CP to SEQ : Control State store load 28~~2424~~
22.1.11 MH to SEQ: Instruction store Load 28~~2425~~
22.1.12 SP0 to SX0 : Pixel read from RBs 28~~2425~~
22.1.13 SEQ to SX0 : Control bus.........29~~2425~~
22.1.14 SX0 to SEQ : Output file control 29~~2425~~
22.1.15 SP0 to SX0 : Position return bus 30~~2526~~
22.1.16 Shader Engine to Fetch Unit Bus (Fast Bus) 30~~2526~~
22.1.17 Sequencer to Fetch Unit bus (Slow Bus) 30~~2526~~
23. EXAMPLES OF PROGRAM EXECUTIONS 31~~2526~~
23.1.1 Sequencer Control of a Vector of Vertices 32~~2626~~
23.1.2 Sequencer Control of a Vector of Pixels 34~~2727~~
23.1.3 Notes ....34~~2828~~
24. OPEN ISSUES..35~~2828~~

Revision Changes:

Rev 0.1 (Laurent Lefebvre) Date: May 7, 2001	First draft.
Rev 0.2 (Laurent Lefebvre) Date : July 9, 2001	Changed the interfaces to reflect the changes in the SP. Added some details in the arbitration section.
Rev 0.3 (Laurent Lefebvre) Date : August 6, 2001	Reviewed the Sequencer spec after the meeting on August 3, 2001.
Rev 0.4 (Laurent Lefebvre) Date : August 24, 2001	Added the dynamic allocation method for register file and an example (written in part by Vic) of the

	flow of pixels/vertices in the sequencer.
Rev 0.5 (Laurent Lefebvre) Date : September 7, 2001	Added timing diagrams (Vic)
Rev 0.6 (Laurent Lefebvre) Date : September 24, 2001	Changed the spec to reflect the new R400 architecture. Added interfaces.
Rev 0.7 (Laurent Lefebvre) Date : October 5, 2001	Added constant store management, instruction store management, control flow management and data dependant predication.
Rev 0.8 (Laurent Lefebvre) Date : October 8, 2001	Changed the control flow method to be more flexible. Also updated the external interfaces.
Rev 0.9 (Laurent Lefebvre) Date : October 17, 2001	Incorporated changes made in the 10/18/01 control flow meeting. Added a NOP instruction, removed the conditional_execute_or_jump. Added debug registers.
Rev 1.0 (Laurent Lefebvre) Date : October 19, 2001	Refined interfaces to RB. Added state registers.
Rev 1.1 (Laurent Lefebvre) Date : October 26, 2001	Added SEQ→SP0 interfaces. Changed delta precision. Changed VGT→SP0 interface. Debug Methods added.
Rev 1.2 (Laurent Lefebvre) Date : November 16, 2001	Interfaces greatly refined. Cleaned up the spec.

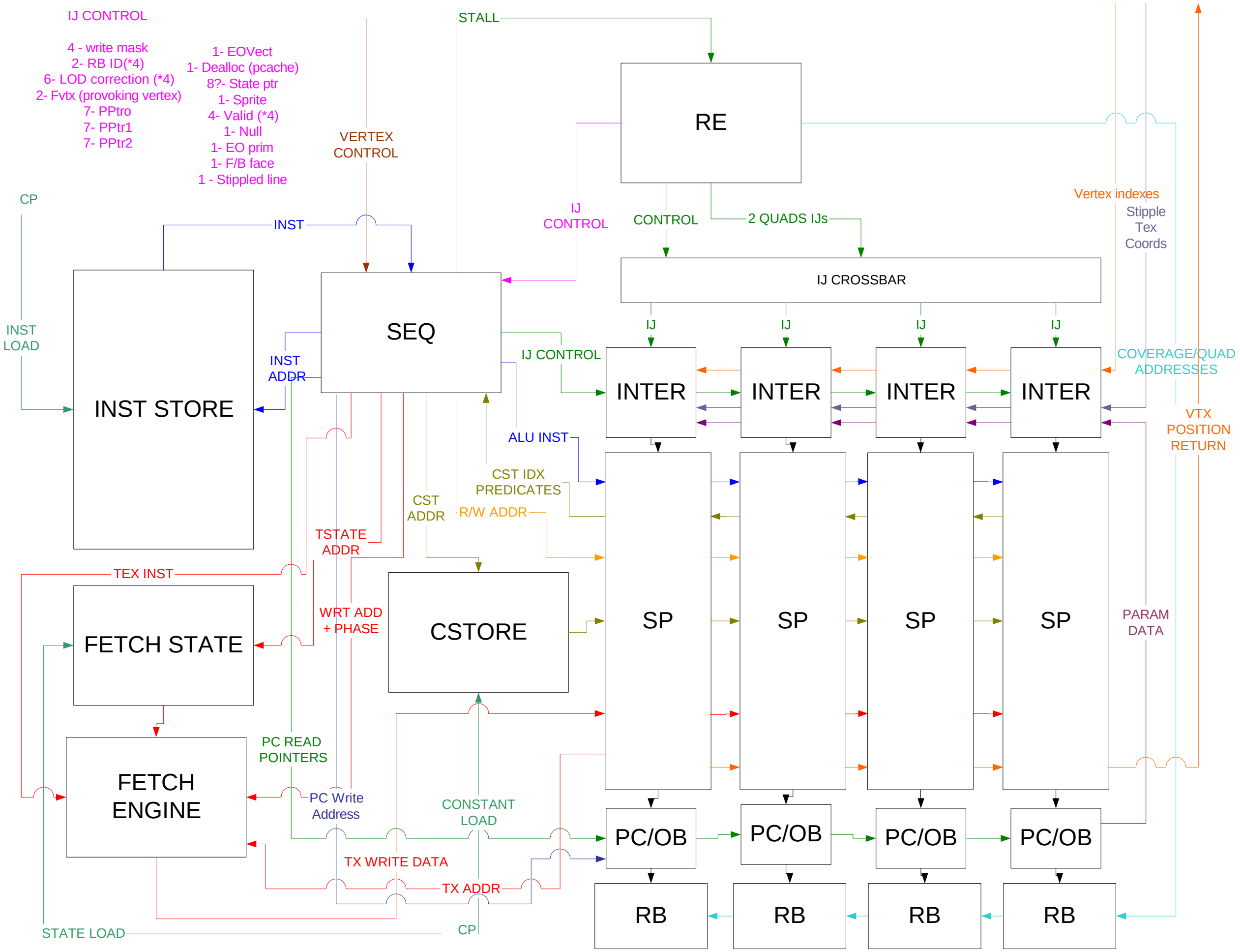
1. Overview

The sequencer first arbitrates between vectors of 64 vertices that arrive directly from primitive assembly and vectors of 16 quads (64 pixels) that are generated in the raster engine.

The vertex or pixel program specifies how many GPR's it needs to execute. The sequencer will not start the next vector until the needed space is available.

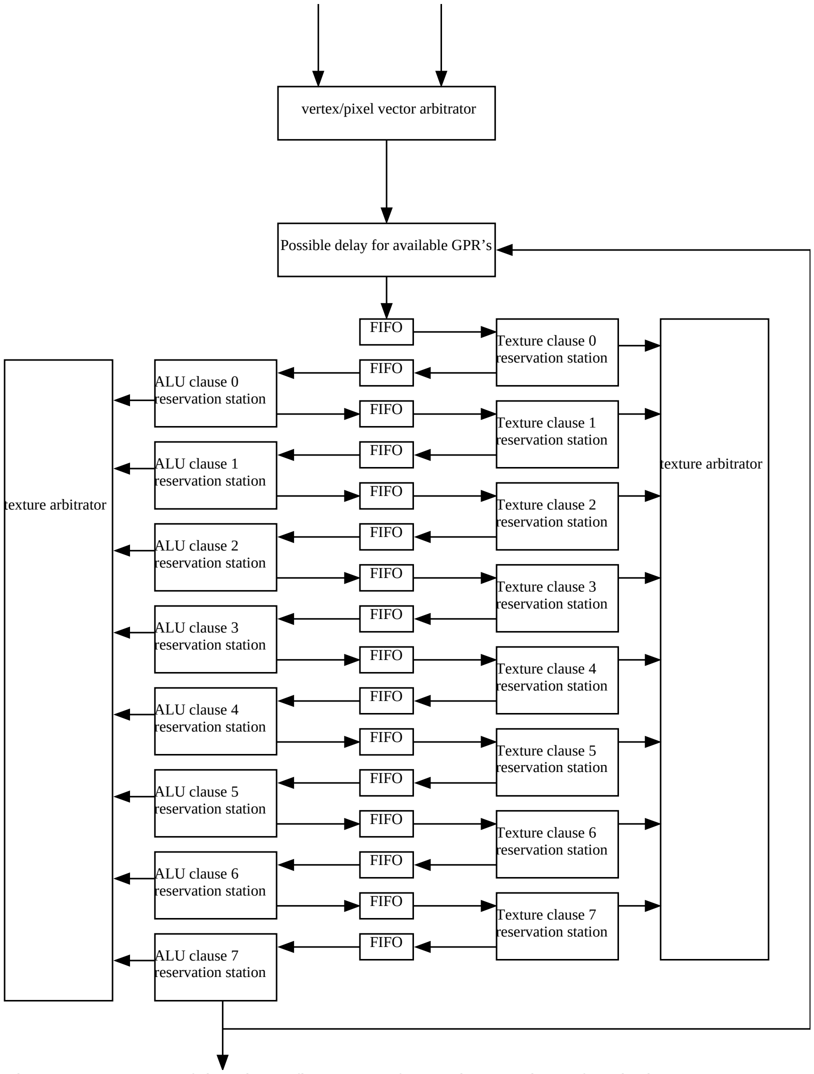
The sequencer is based on the R300 design. It chooses two ALU clauses and a fetch clause to execute, and executes all of the instructions in a clause before looking for a new clause of the same type. Two ALU clauses are executed interleaved to hide the ALU latency. Each vector will have eight fetch and eight ALU clauses, but clauses do not need to contain instructions. A vector of pixels or vertices ping-pongs along the sequencer FIFO, bouncing from fetch reservation station to alu reservation station. A FIFO exists between each reservation stage, holding up vectors until the vector currently occupying a reservation station has left. A vector at a reservation station can be chosen to execute. The sequencer looks at all eight alu reservation stations to choose an alu clause to execute and all eight fetch stations to choose a fetch clause to execute. The arbitrator will give priority to clauses/reservation stations closer to the bottom of the pipeline. It will not execute an alu clause until the fetch fetches initiated by the previous fetch clause have completed. There are two separate sets of reservation stations, one for pixel vectors and one for vertices vectors. This way a pixel can pass a vertex and a vertex can pass a pixel.

To support the shader pipe the raster engine also contains the shader instruction cache and constant store. There are only one constant store for the whole chip and one instruction store. These will be shared among the four shader pipes. The four shader pipes also execute the same ~~instuction~~instruction thus there is only one sequencer for the whole chip.



IJ CONTROL
4 - write mask
2- RB ID(*4)
6- LOD correction (*4)
2- Fvtx (provoking vertex)
7- PPtro
7- PPtr1
7- PPtr2
1- EOVect
1- Dealloc (pcache)
8?- State ptr
1- Sprite
4- Valid (*4)
1- Null
1- EO prim
1- F/B face
1 - Stippled line
VERTEX CONTROL
STALL
RE
CP
INST
IJ CONTROL
CONTROL
2 QUADS IJs
Vertex indexes
Stipple Tex Coords
IJ CROSSBAR
INST LOAD
INST STORE
SEQ
INST ADDR
IJ CONTROL
IJ
IJ
IJ
IJ
INTER
INTER
INTER
INTER
COVERAGE/QUAD ADDRESSES
VTX POSITION RETURN
ALU INST
CST IDX PREDICATES
CST ADDR
R/W ADDR
TSTATE ADDR
TEX INST
SP
SP
SP
SP
PARAM DATA
FETCH STATE
WRT ADD + PHASE
CSTORE
PC READ POINTERS
FETCH ENGINE
PC Write Address
CONSTANT LOAD
PC/OB
PC/OB
PC/OB
PC/OB
TX WRITE DATA
TX ADDR
RB
RB
RB
RB
STATE LOAD
CP

1.1 Top Level Block Diagram



vertex/pixel vector arbitrator
Possible delay for available GPR's
FIFO
Texture clause 0 reservation station
Texture clause 1 reservation station
Texture clause 2 reservation station
Texture clause 3 reservation station
Texture clause 4 reservation station
Texture clause 5 reservation station
Texture clause 6 reservation station
Texture clause 7 reservation station
texture arbitrator
ALU clause 0 reservation station
ALU clause 1 reservation station
ALU clause 2 reservation station
ALU clause 3 reservation station
ALU clause 4 reservation station
ALU clause 5 reservation station
ALU clause 6 reservation station
ALU clause 7 reservation station
texture arbitrator

There are two sets of the above figure, one for vertices and one for pixels.

Depending on the arbitration state, the sequencer will either choose a vertex or a pixel packet. The control packet consists of ~~21~~3 bits of state, 6-7 bits for the base address of the Shader program and some information on the coverage to determine fetch LOD plus other various small state bits.

On receipt of a packet, the input state machine (not pictured but just before the first FIFO) allocated enough space in the registers to store the interpolated values and temporaries. Following this, the input state machine stacks the packet in the first FIFO.

On receipt of a command, the level 0 fetch machine issues a ~~texure~~texture request and corresponding register address for the fetch address (ta). A small command (tcmd) is passed to the fetch system identifying the current level number (0) as well as the register write address for the fetch return data. One fetch request is sent every 4 clocks causing the texturing of sixteen 2x2s worth of data (or 64 vertices). Once all the requests are sent the packet is put in FIFO 1.

Upon recept of the return data, the fetch unit writes the data to the register file using the write address that was provided by the level 0 fetch machine and sends the clause number (0) to the level 0 fetch state machine to signify that the write is done and thus the data is ready. Then, the level 0 fetch machine increments the counter of FIFO 1 to signify to the ALU 1 that the data is ready to be processed.

On receipt of a command, the level 0 ALU machine first decrements the input FIFO counter and then issues a complete set of level 0 shader instructions. For each instruction, the state machine generates 3 source addresses, one destination address (3 cycles later) and an instruction. Once the last instruction as been issued, the packet is put into FIFO 2.

There will always be two active ALU clauses at any given time (and two ~~arbitrers~~arbiters). One ~~arbitrer~~arbiter will arbitrate over the odd instructions (4 clocks cycles) and the other one will arbitrate over the even instructions (4 clocks cycles). The only constraints between the two ~~arbitrers~~arbiters is that they are not allowed to pick the same clause number as the other one is currently working on if the packet is not of the same type (render state).

If the packet is a vertex packet, upon reaching ALU clause 3, it can export the position if the position is ready. So the ~~arbitrer~~arbiter must prevent ALU clause 3 to be selected if the positional buffer is full (or can't be accessed). Along with the positional data, the location where the vertex data is to be put is also sent (parameter data pointers).

{ISSUE: How do we handle parameter cache pointers (computed, semi-computed or not computed)?}

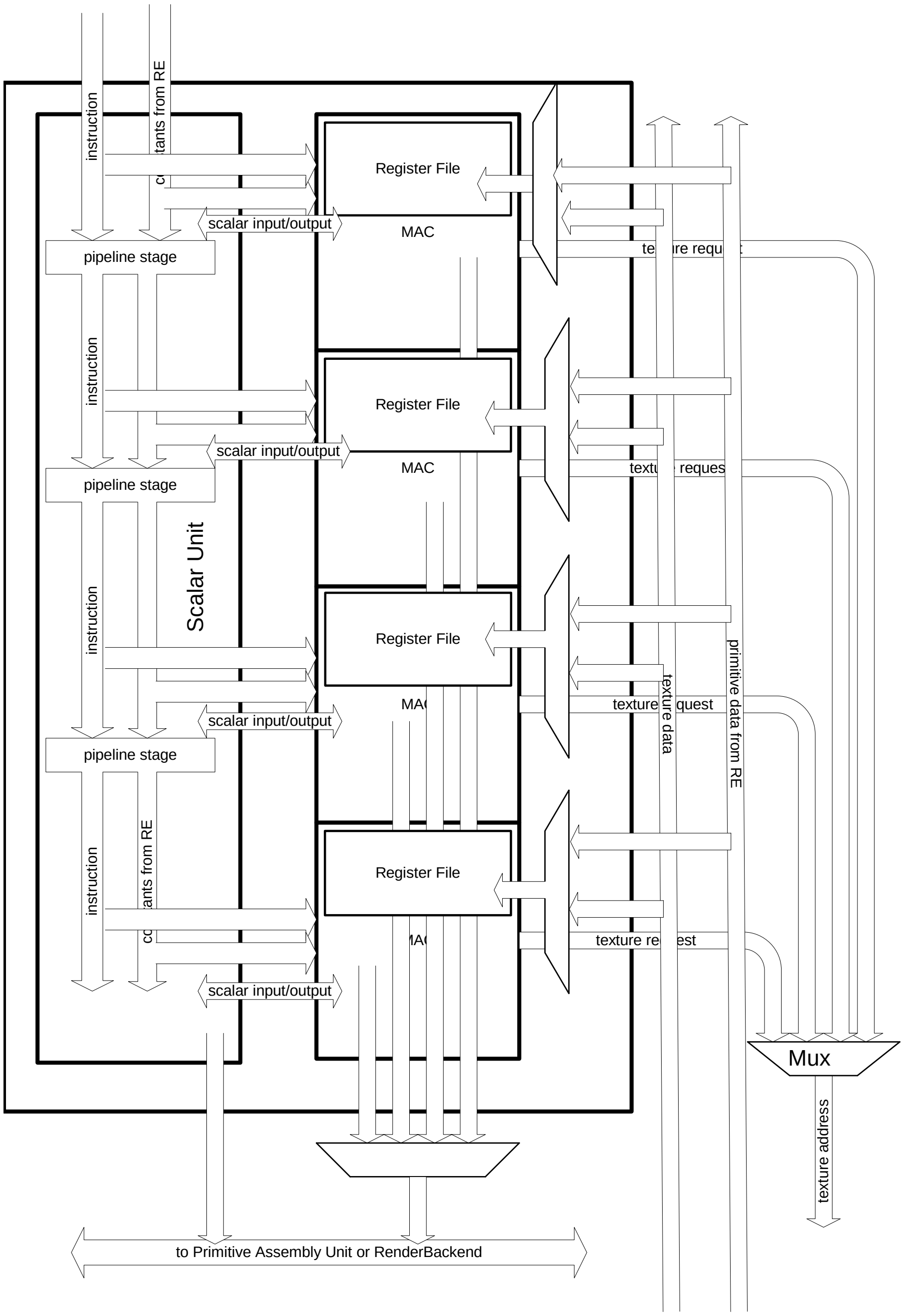
A special case is for HOS surfaces wich can export 12 parameters per last 6 clauses to the output buffer. If the output buffer is full or doesn't have enough space the sequencer will prevent such a vertex group to enter an exporting clause.

Regular pixel and vertex shaders can export 12 parameters to memory from the last clause only (7).

All other level process in the same way until the packet finally reaches the last ALU machine (7). On completion of the level 7 ALU clause, a valid bit is sent to the Render Backend which picks up the color data. This requires that the last instruction writes to the output register – a condition that is almost always true. If the packet was a vertex packet, instead of sending the valid bit to the RB, it is sent to the PA so it can know that the data present in the parameter store is valid.

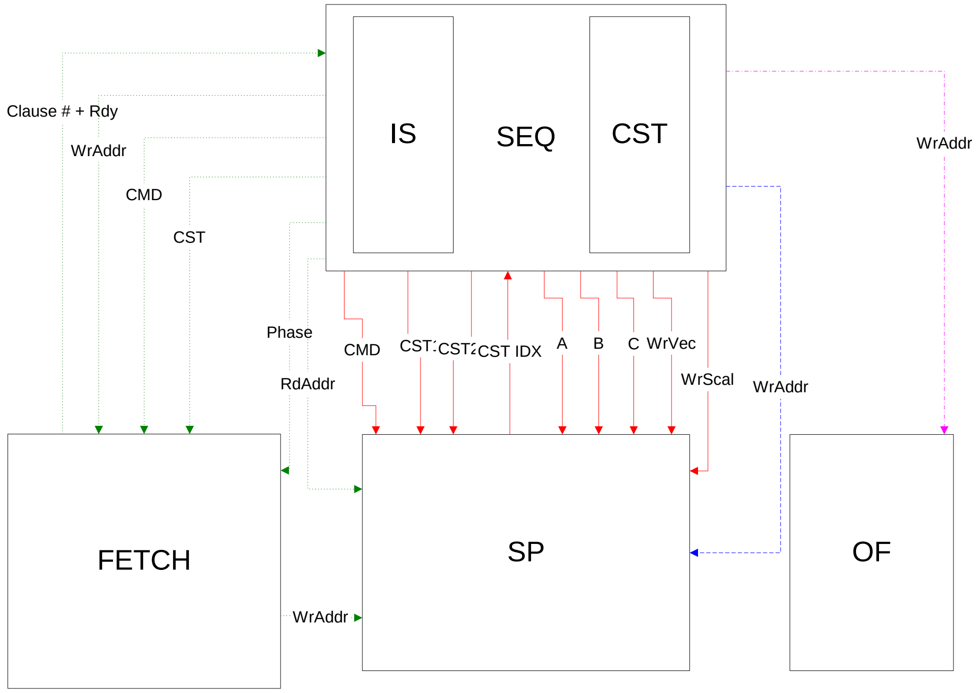
Only two ALU state machine may have access to the register file address bus or the instruction decode bus at one time. Similarly, only one fetch state machine may have access to the register file address bus at one time. Arbitration is performed by three ~~arbitrer~~arbiter blocks (two for the ALU state machines and one for the fetch state machines). The ~~arbitrers~~arbiters always favor the higher number state machines, preventing a bunch of half finished jobs from clogging up the register files.

1.2 Data Flow graph



The gray area represents blocks that are replicated 4 times per shader pipe (16 times on the overall chip).

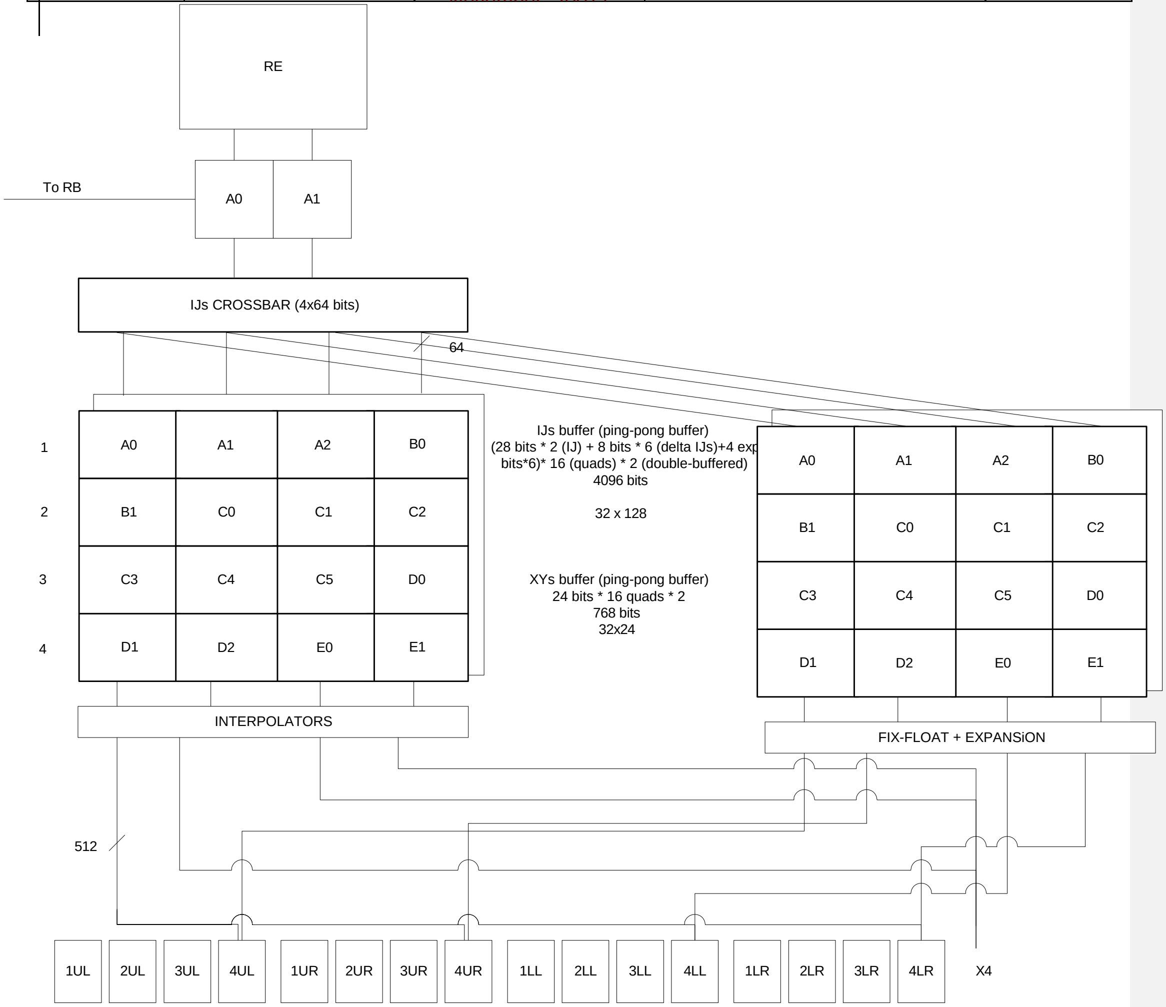
1.3 Control Graph



In green is represented the Fetch control interface, in red the ALU control interface, in blue the Interpolated/Vector control interface and in purple is the output file control interface.

2. Interpolated data bus

The interpolators contain an IJ buffer to pack the information as much as possible before writing it to the register file.



IJs buffer (ping-pong buffer)
(28 bits * 2 (IJ) + 8 bits * 6 (delta IJs)+4 exp
bits*6)* 16 (quads) * 2 (double-buffered)
4096 bits

32 x 128

XYs buffer (ping-pong buffer)
24 bits * 16 quads * 2
768 bits
32x24

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21	T22	T23
SP 0	A0	B1	C3	D1					A0	B1	C3	D1					XY 0-3	XY 16-19	XY 32-35	XY 48-51	V 0-3	V 16-19	V 32-35	V 48-51
SP 1	A1		C4	D2		C0			A1		C4	D2		C0			XY 4-7	XY 20-23	XY 36-39	XY 52-55	V 4-7	V 20-23	V 36-39	V 52-55
SP 2	A2		C5			C1		E0	A2		C5			C1		E0	XY 8-11	XY 24-27	XY 40-43	XY 56-59	V 8-11	V 24-27	V 40-43	V 56-59
SP 3					B0	C2	D0	E1					B0	C2	D0	E1	XY 12-15	XY 28-31	XY 44-47	XY 60-63	V 12-15	V 28-31	V 44-47	V 60-63
				P0								P1						XY				VTX		

Above is an example of a tile we might receive. The IJ information is packed in the IJ buffer 2 quads at a time. The sequencer allows at any given time as many as four quads to interpolate a parameter. They all have to come from the same primitive. Then the sequencer controls the write mask to the register to write the valid data in.

3. Instruction Store

There is going to be only one instruction store for the whole chip. It will contain 4096 instructions of 96 bits each. ~~There is also going to be a control instruction store of size 256(512?)x32.~~

~~{ISSUE : The instruction store is loaded by the sequencer using the memory hub ?}.~~

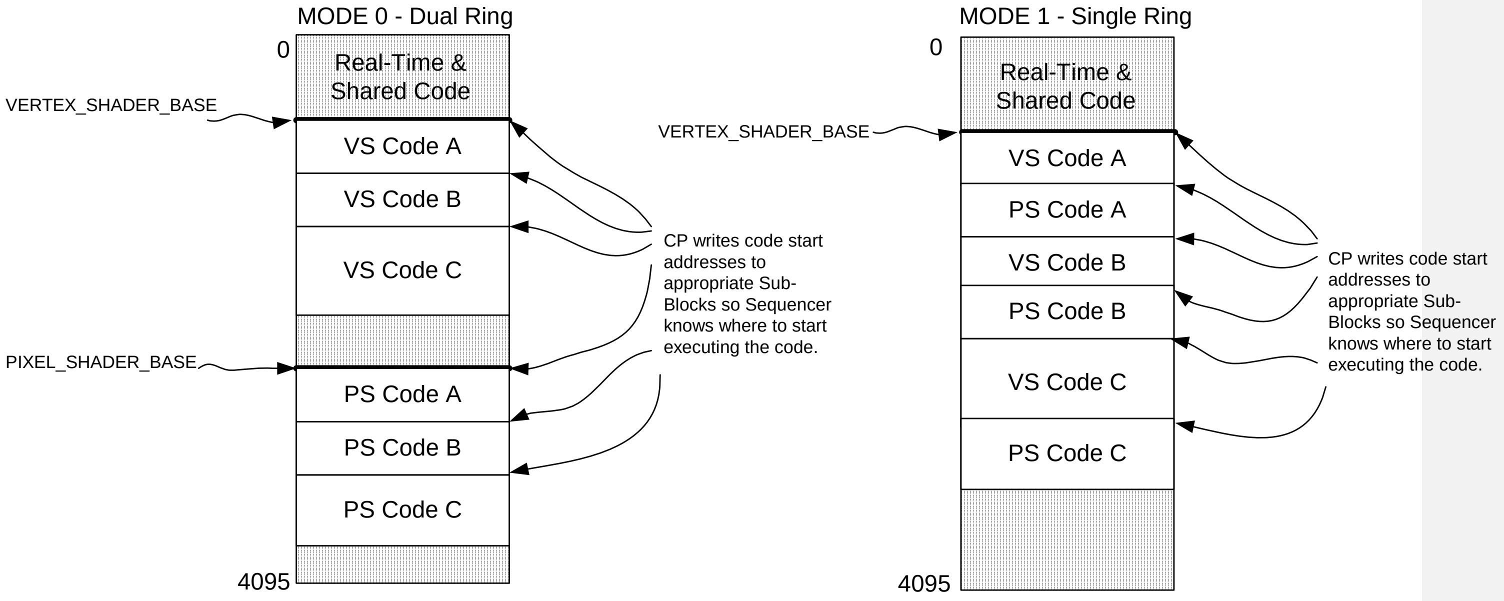
~~The read bandwith from this store is 96*2 bits/ 4 clocks (48 bits/clock).~~ It is likely to be a 1 port memory; we use 1 clock to load the ALU instruction, 1 clocks to load the Fetch instruction, 1 clock to load 2 control flow instructions and 1 clock to write instructions.

The instruction store is loaded by the CP thru the INSTRUCTION_DATA, INSTRUCTION_INDEX_PORT control registers. The INSTRUCTION_INDEX_PORT is auto-incremented on both reads and writes to the INSTRUCTION_DATA register.

The next picture shows the various modes the CP can load the memory. The Sequencer has to keep track of the loading modes in order to wrap around the correct boundaries. The MSB of the INSTRUCTION_INDEX_PORT register contains the packet type for the sequencer to know where it must wrap around. The wrap around points are arbitrary and they are specified in the VERTEX_SHADER_BASE and PIXEL_SHADER_BASE registers.

R400 CP's Views of Instruction Memory

Updated: 11/14/2001
John A. Carey



CP writes code start addresses to appropriate Sub-Blocks so Sequencer knows where to start executing the code.

CP writes code start addresses to appropriate Sub-Blocks so Sequencer knows where to start executing the code.

4. Sequencer Instructions

All control flow instructions and move instructions are handled by the sequencer only. The ALUs will perform NOPs during this time (MOV PV,PV, PS,PS).

5. Constant Stores

~~The constant store is managed by the CP.~~ The sequencer is aware of where the constants are using a remaping table~~ also managed by the CP~~. A likely size for the constant store is ~~512x128~~ 1024x128 bits. ~~The constant store is also planned to be shared.~~ The read BW from the constant store is 128 bits/clock and the write bandwith is 32/4 bits/clock.

In order to do constant store indexing, the sequencer must be loaded first with the indexes (that come from the GPRs). There are 144 wires from the exit of the SP to the sequencer (9 bits pointers x 16 vertexes/clock). Since the data must pass thru the Shader pipe for the float to fixed convertion, there is a latency of 4 clocks (1 instruction) between the time the sequencer is loaded and the time one can index into the constant store. The assembly will look like this

```
MOVA  R1.X,R2.X       // Loads the sequencer with the content of R2.X, also copies the content of R2.X into R1.X
NOP                   // latency of the float to fixed conversion
ADD   R3,R4,C0[R2.X]// Uses the state from the sequencer to add R4 to C0[R2.X] into R3
```

Note that we don't really care about what is in the brackets because we use the state from the MOVA instruction. R2.X is just written again for the sake of simplicity.

The storage needed in the sequencer in order to support this feature is 2*64*9 bits = 1152 bits.

The texture state is also kept in a similar memory. The size of this memory is 192x128. Which lets us load a texture state in ????

The control flow constant memory doesn't sit behind a renaming table. It is register mapped and thus the driver must reload its content each time there is a state change.

6. Looping and Branches

Loops and branches are planned to be supported and will have to be dealt with at the sequencer level. We plan on supporting constant loops and branches using a control program.

6.1 The controlling state.

As per Dx9 the following state is available for control flow:

Boolean[15:0]
loop_count[7:0][7:0]
In addition:
loop_start [7:0] [7:0]
loop_step [7:0] [7:0]
Exist to give more control to the controlling program.

We will extend that in the R400 to:
Boolean[255:0]
Loop_count[7:0][15:0]
Loop_Start[7:0] [15:0] times ~~2~~ 3 (one for constant,registert1, register2)
Loop_Step[7:0] [15:0] times ~~2~~ 3 (one for constant, registert1, register2~~register~~)
Loop_End[7:0] [15:0]

{ISSUE: How is the controlling state loaded and how many contexts do we have?}

We have a stack of 4 elements for calling subroutines and 4 loop counters to allow for nested loops.

6.2 The Control Flow Program

The R300 uses a match method for control flow: The shader is executed, and at every instruction its address is compared with addresses (or address?) in a control table. The “event” in the control table can redirect operations in the program.

The Method chosen for the R400 is a “control program”. The control program has ten basic instructions:

Execute
Conditional_execute
Conditional_Execute_Predicates
Conditional_jump
Call
Return
Loop_start
Loop_end
End_of_clause
NOP

Execute, causes the specified number of instructions in instruction store to be executed.
Conditional_execute checks a condition first, and if true, causes the specified number of instructions in instruction store to be executed.
Loop_start resets the corresponding loop counter to the start value on the first pass after it checks for the end condition and if met jumps over to a specified address.
Loop_end increments (decrements?) the loop counter and jumps back the specified number of instructions.
Call jumps to an address and pushes the IP counter on the stack. On the return instruction, the IP is poped from the stack.
Conditional_execute_or_Jump executes a block of instructions or jumps to an address is the condition is not met.
Conditional_execute_Predicates executes a block of instructions if all bits in the predicate vectors meet the condition.
End_of_clause marks the end of a clause.
Conditional_jumps jumps to an address if the condition is met.
NOP is a regular NOP

NOTE THAT ALL JUMPS MUST JUMP TO EVEN CFP ADDRESSES. Thus the compiler must insert NOPs where needed to align the jumps on even CFP addresses.

Also if the jump is logically bigger than pshader_cntl_size (or vshader_cntl_size) we break the program (clause) and set the debug registers. If an execute or conditional_execute is lower than cntl_size or bigger than size we also break the program (clause) and set the debug registers.

We have to fit instructions into 48 bits in order to be able to put two control flow instruction per line in the instruction store.

Execute				
47	46… 42	41 … 24	23 … 12	11 … 0
Addressing	00001	RESERVED	Instruction _count	Exec Address

Execute up to 4k instructions at the specified address in the instruction memory.

NOP		
47	46 … 42	41 … 0
Addressing	00010	RESERVED

~~If the specified boolean (8 bits can address 256 booleans) meets the specified condition then execute the specified instructions (up to 512 instructions) or if the condition is not met jump to the jump address in the control flow program. This MUST be a forward jump.~~This is a regular NOP.

Conditionnal_Execute						
47	46 … 42	41 … 34	33	32 … 24	23 … 12	11 … 0
Addressing	00011	Boolean address	Condition	RESERVED	Instruction_count	Exec Address

If the specified boolean (8 bits can address 256 booleans) meets the specified condition then execute the specified instructions (up to 4k instructions)

Conditionnal_Execute_Predicates						
47	46 … 42	41 … 38	37	36 … 24	23 … 12	11 … 0
Addressing	00100	Predicate vector	Condition	RESERVED	Instruction_count	Exec Address

Check the AND/OR of all current predicate bits. If AND/OR matches the condition execute the specified number of instructions. We need to AND/OR this with the kill mask in order not to consider the pixels that aren't valid.

Loop_Start				
47	46 … 42	41 … 16	15 … 4	3 … 0
Addressing	00101	RESERVED	Jump address	Loop ID

Loop Start. Compares the loop count with the end value. If loop condition not met jump to the address. Forward jump only. Also computes the index value.

Loop_End				
47	46 … 42	41 … 16	15 … 4	3 … 0
Addressing	00111	RESERVED	Start address	Loop ID

Loop end. Increments the counter by one and jumps BACK only to the start of the loop.

The way this is described does not prevent nested loops, and the inclusion of the loop id make this easy to do.

Call			
47	46 … 42	41…12	11 … 0
Addressing	01000	RESERVED	Address

Jumps to the specified address and pushes the IP counter on the stack.

Return		
47	46 … 42	41 … 0
Addressing	01001	RESERVED

Pops the topmost address from the stack and jumps to that address. If nothing is on the stack, the program will just continue to the next instruction.

Conditionnal_Jump						
47	46 … 42	41 … 34	33	32 … 13	12	11 … 0
Addressing	01010	Boolean address	Condition	RESERVED	FW only	Address

If condition met, jumps to the address. FORWARD jump only allowed if bit 12 set. Bit 12 is only an optimization for the compiler and should NOT be exposed to the API.

End_of_Clause		
47	46 … 42	41 … 0
Addressing	01011	RESERVED

Marks the end of a clause.

To prevent infinite loops, we will keep 9 bits loop counters instead of 8 (we are only able to loop 256 times). If the counter goes higher than 255 then the loop_end or the loop_start instruction is going to break the loop and set th~~d~~e debug registers. ~~The sequencer will keep two loop indexes values:~~

~~IC index for constant indexing (9 bits)~~

~~IR index for register file indexing (7 bits)~~

~~This will be updated every time we loop and can only be used to index the constant store and the register file. The way to compute this value is:~~

~~Index = Loop_counter*Loop_iterator + Loop_init.~~

~~The IC for constant is going to return 0 if it is out of the constant range. The IR index is going to break the program if the index exceeds the number of requested registers.~~

The basic model is as follows:

The render state defined the clause boundaries:

Vertex_shader_fetch[7:0][7:0]	// eight 8 bit pointers to the location where each clauses control program is located
Vertex_shader_alu[7:0][7:0]	// eight 8 bit pointers to the location where each clauses control program is located
Pixel_shader_fetch[7:0][7:0]	// eight 8 bit pointers to the location where each clauses control program is located
Pixel_shader_alu[7:0][7:0]	// eight 8 bit pointers to the location where each clauses control program is located

A pointer value of FF means that the clause doesn't contain any instructions.

Formatted

The control program for a given clause is executed to completion before moving to another clause, (with the exception of the pick two nature of the alu execution). The control program is the only program aware of the clause boundaries.

6.3 Data dependant predicate instructions

Data dependant conditionals will be supported in the R400. The only way we plan to support those is by supporting three vector/scalar predicate operations of the form:

PRED_SETE_# - similar to SETE except that the result is 'exported' to the sequencer.
PRED_SETGT_# - similar to SETGT except that the result is 'exported' to the sequencer
PRED_SETGTE_# - similar to SETGTE except that the result is 'exported' to the sequencer

For the scalar operations only we will also support the two following instructions:
PRED_SETE0_# – SETE0
PRED_SETE1_# – SETE1

The export is a single bit - 1 or 0 that is sent using the same data path as the MOVA instruction. The sequencer will maintain 4 sets of 64 bit predicate vectors (in fact 8 sets because we interleave two programs but only 4 will be exposed) and use it to control the write masking. This predicate is not maintained across clause boundaries. The # sign is used to specify which predicate set you want to use 0 thru 3.

Then we have two conditional execute bits. The first bit is a conditional execute "on" bit and the second bit tells us if we execute on 1 or 0. For example, the instruction:

P0_ADD_# R0,R1,R2

Is only going to write the result of the ADD into those GPRs whose predicate bit is 0. Alternatively, P1_ADD_# would only write the results to the GPRs whose predicate bit is set. The use of the P0 or P1 without precharging the sequencer with a PRED instruction is undefined.

{Issue: do we have to have a NOP between PRED and the first instruction that uses a predicate?}

6.4 HW Detection of PV,PS

Because of the control program, the compiler cannot detect statically dependant instructions. In the case of non-masked writes and subsequent reads the sequencer will insert uses of PV,PS as needed. This will be done by comparing the read address and the write address of consecutive instructions. For masked writes, the sequencer will insert NOPs wherever there is a dependant read/write.

The sequencer will also have to insert NOPs between PRED_SET and MOVA instructions and their uses.

6.5 Register file indexing

Because we can have loops in fetch clause, we need to be able to index into the register file in order to retrieve the data created in a fetch clause loop and use it into an ALU clause. The instruction will include the base address for register indexing and the instruction will contain these controls:

Bit7	Bit 6	
0	0	'absolute register'
0	1	'relative register'
1	0	'previous vector'
1	1	'previous scalar'

In the case of an absolute register we just take the address as is. In the case of a relative register read we take the base address and we add to it the loop_index and this becomes our new address that we give to the shader pipe.

The sequencer is going to keep a loop index computed as such:

Index = Loop_counter*Loop_iterator + Loop_init.

The index is going to return 0 if it is out of the range.

6.6 Predicated Instruction support for Texture clauses

For texture clauses, we support the following optimization: we keep 1 bit (thus 4 bits for the four predicate vectors) per predicate vector in the reservation stations. A value of 1 means that one ore more elements in the vector have a value of one (thus we have to do the texture fetches for the whole vector. A value of 0 means that no elements in the vector have his predicate bit set and we can thus skip over the texture fetch. **We have to make sure the invalid pixels aren't considered with this optimization.**

Formatted

6.7 Debugging the Shaders

In order to be able to debug the pixel/vertex shaders efficiently, we provide ~~3~~ 2 methods.

6.7.1 *Method 1: Debugging registers*

Current plans are to expose 2 debugging, or error notification, registers:
1. address register where the first error occurred
2. count of the number of errors

The sequencer will detect the following groups of errors:
- count overflow
- jump error
  relative jump address > size of the control flow program
  relative jump address > length of the shader program

- constant overflow
- register overflow
- call stack
  call with stack full
  return with stack empty

With two of the errors, a jump error or a register overflow will cause the program to break. In this case, a break means that a clause will halt execution, but allowing further clauses to be executed.

With the other errors, program can continue to run, potentially to worst-case limits.

If indexing outside of the constant range, causing an overflow error, the hardware is specified to return the value with an index of 0. This could be exploited to generate error tokens, by reserving and initializing the 0th register (or constant) for errors.

{ISSUE : Interrupt to the driver or not?}

6.7.2 *Method 2: Exporting the values in the GPRs (12)*

The sequencer will have a count register and an address register for this mode and 3 bits per clause specifying the execution mode for each clause. The modes can be :

1) Normal
2) Debug Kill
3) Debug Addr + Count

~~4)Debug Count~~

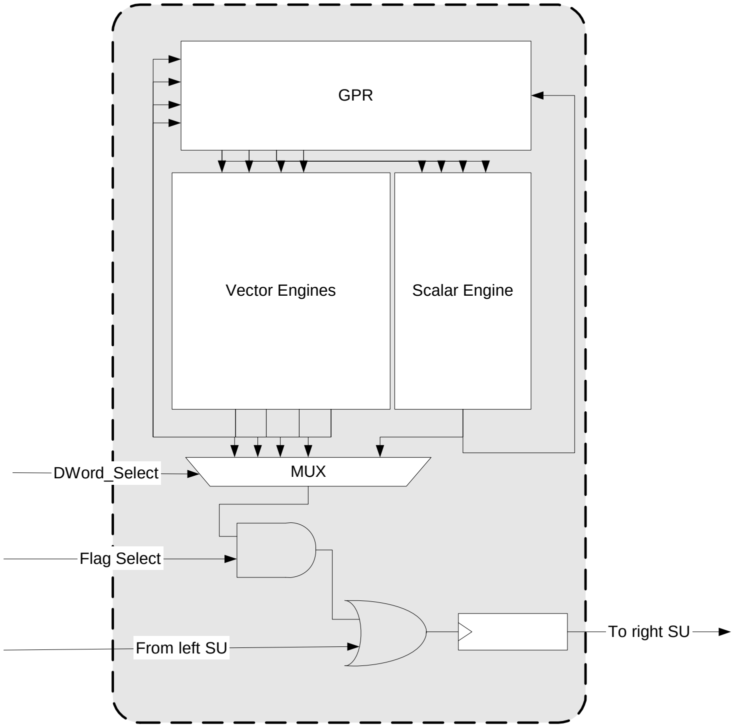
Formatted: Bullets and Numbering

Under the normal mode execution follows the normal course. Under the kill mode, all control flow instructions are executed but all normal shader instructions of the clause are replaced by NOPs. Only debug_export instructions of clause 7 will be executed under the debug kill setting. Under the ~~two~~ other mode~~s~~, normal execution is done until we reach an address specified by the address register ~~or~~ and instruction count (useful for loops) specified by the count register. ~~After we have hit the address or the count we change to the kill mode for the rest of the clause~~ After we have hit the instruction n times (n=count) we switch the clause to the kill mode.~~.~~

~~6.7.3 Method 3: Selective export of a 32 bit Dword.~~

Formatted: Bullets and Numbering

~~The third debug option will be mainly used for HW debug. For this mode, the sequencer will keep the following control debug registers: Shader_pipe (6 bits), Mode(1 bit), Dword_select (3 bits), clause_+count (16 bits?),address (12 bits) Vector_number (8 bits), Render_state (21 bits). The shader pipe register selects a shader pipe amongst the 64, the dword_select selects a channel (0…3 of vector or scalar), the clause_+count selects at which clause and which count we export, the Render_state specifies which render state is concerned, the Vector_number specifies which vector is concerned and the mode selects count export, or address export.~~



~~Flag Select is a combination of Shader_pipe, clause_+count, address, Vector_number and render_state. It is only active for 1 shader pipe at a time and for 1 vector of a given state. The driver is responsible to reset the output register to 0 before executing a given program.~~

7. Pixel Kill Mask

A vector of 64 bits is kept per group of pixels/vertices. Its purpose is to optimize the texture fetch requests and allow the shader pipe to kill pixels using the following instructions:

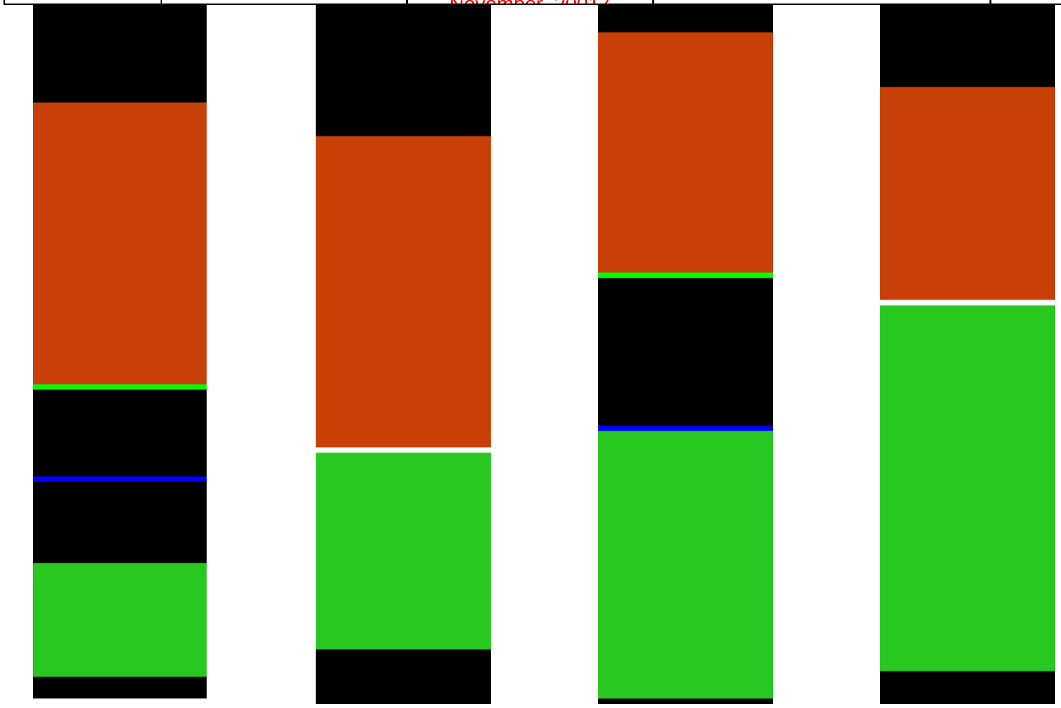
MASK_SETE
MASK_SETGT
MASK_SETGTE

8. HOS surfaces

HOS surfaces are able to export from the 6 last clauses but to memory ONLY. If they want to export to the parameter cache they have to do it in the last clause (7). They can also export position in clause 3.

9. Register file allocation

The register file allocation for vertices and pixels can either be static or dynamic. In both cases, the register file in managed using two round robins (one for pixels and one for vertices). In the dynamic case the boundary between pixels and vertices is allowed to move, in the static case it is fixed to VERTEX_REG_SIZE for vertices and 256-VERTEX_REG_SIZE for pixels.



Above is an example of how the algorithm works. Vertices come in from top to bottom; pixels come in from bottom to top. Vertices are in orange and pixels in green. The blue line is the tail of the vertices and the green line is the tail of the pixels. Thus anything between the two lines is shared. When pixels meets vertices the line turns white and the boundary is static until both vertices and pixels share the same "unallocated bubble". Then the boundary is allowed to move again.

10. Fetch Arbitration

The fetch arbitration logic chooses one of the 8 potentially pending fetch clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute. Once chosen, the clause state machine will send one 2x2 fetch per clock (or 4 fetches in one clock every 4 clocks) until all the fetch instructions of the clause are sent. This means that there cannot be any dependencies between two fetches of the same clause.

The arbitrator will not wait for the fetches to return prior to selecting another clause for execution. The fetch pipe will be able to handle up to X(?) in flight fetches and thus there can be a fair number of active clauses waiting for their fetch return data.

11. ALU Arbitration

ALU arbitration proceeds in almost the same way than fetch arbitration. The ALU arbitration logic chooses one of the 8 potentially pending ALU clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute. There are two ALU arbitrers, one for the even clocks and one for the odd clocks. For exemple, here is the sequencing of two interleaved ALU clauses (E and O stands for Even and Odd sets of 4 clocks):

Einst0 Oinst0 Einst1 Oinst1 Einst2 Oinst2 Einst0 Oinst3 Einst1 Oinst4 Einst2 Oinst0…
Proceeding this way hides the latency of 8 clocks of the ALUs.

12. Handling Stalls

When the output file is full, the sequencer prevents the ALU arbitration logic to select the last clause (this way nothing can exit the shader pipe until there is place in the output file. If the packet is a vertex packet and the position buffer is full (POS_FULL) then the sequencer also prevents a thread to enter the exporting clause (43?). The sequencer will set the OUT_FILE_FULL signal n clocks before the output file is actually full and thus the ALU ~~arbitrer~~arbiter will be able read this signal and act accordingly by not preventing exporting clauses to proceed.

13. Content of the reservation station FIFOs

21 bits of Render State 7 bits for the base address of the GPRs, some bits for LOD correction and coverage mask information in order to fetch fetch for only valid pixels, quad address and 1 bit to specify if the vector is of pixels or vertices. Since pixels and vertices are kept in order in the shader pipe, we only need two fifos (one for vertices and one for pixels) deep enough to cover the shader pipe latency. This size will be determined later when we will know the size of the small fifos between the reservation stations.

14. The Output File

The output file is where pixels are put before they go to the RBs. The write BW to this store is 256 bits/clock. Just before this output file are staging registers with write BW 512 bits/clock and read BW 256 bits/clock. ~~For this reason only ONE concurrent program can be of clause 8 (exporting clause) the other program MUST not.~~ The staging registers are 4x128 (and there are 16 of those on the whole chip).

15. IJ Format

The IJ information sent by the PA is of this format on a per quad basis:

We have a vector of IJ's (one IJ per pixel at the centroid of the fragment or at the center of the pixel depending on the mode bit). The interpolation is done at a different precision across the 2x2. The upper left pixel's parameters are always interpolated at full 20x24 mantissa precision. Then the result of the interpolation along with the difference in IJ in reduced precision is used to interpolate the parameter for the other three pixels of the 2x2. Here is how we do it:

Assuming P0 is the interpolated parameter at Pixel 0 having the barycentric coordinates I(0), J(0) and so on for P1,P2 and P3. Also assuming that A is the parameter value at V0 (interpolated with I), B is the parameter value at V1 (interpolated with J) and C is the parameter value at V2 (interpolated with (1-I-J).

$$\begin{aligned}
\Delta 01I &= I(1) - I(0)\\
\Delta 01J &= J(1) - J(0)\\
\Delta 02I &= I(2) - I(0)\\
\Delta 02J &= J(2) - J(0)\\
\Delta 03I &= I(3) - I(0)\\
\Delta 03J &= J(3) - J(0)
\end{aligned}$$

P0	P1
P2	P3

$$\begin{aligned}
P0 &= C + I(0)*(A-C) + J(0)*(B-C)\\
P1 &= P0 + \Delta 01I*(A-C) + \Delta 01J*(B-C)\\
P2 &= P0 + \Delta 02I*(A-C) + \Delta 02J*(B-C)\\
P3 &= P0 + \Delta 03I*(A-C) + \Delta 03J*(B-C)
\end{aligned}$$

P0 is computed at 20x24 mantissa precision and P1 to P3 are computed at 8X24 mantissa precision. So far no visual degradation of the image was seen using this scheme.

Multiplies (Full Precision): 2

Multiplies (Reduced precision): 6
Subtracts 19x24 (Parameters): 2
Adds: 8

FORMAT OF P0's IJ : Mantissa 20 Exp 4 for I + Sign
Mantissa 20 Exp 4 for J + Sign

FORMAT of Deltas (x3):Mantissa 8 Exp 4 for I + Sign
Mantissa 8 Exp 4 for J + Sign

Total number of bits : 19*2 + 8*6 + 4*8 + 4*2 = 128

The Deltas have a leading 1, the Full precision IJs don't. This means that in the case of the deltas we MUST be able to shift 8 right (exponent value of 0 means number = 0, exponent value of 1 means shift right 8). This means that the maximum range for the IJs (Full precision) is +/- 64 and the range for the Deltas is +/- 128.

16. The parameter cache

The parameter cache is where the vertex shaders export their data. It consists of 16 128x128 memories (1R/1W). The reuse engine will make it so that all vertexes of a given primitive will hit different memories.

17. Vertex position exporting

On clause ~~4 (or 5)~~3 the vertex shader can export to the PA both the vertex position and the point sprite. It can also do so at clause ~~8~~ 7 if not done at clause ~~4~~3. ~~Along with the position is exported a pointer to the parameter cache where the data will be once the vertex shader exports.~~ The storage needed to perform the position export is at least 64x128 memories for the position and 64x32 memories for the sprite size. It is going to be taken in the pixel output fifo from the SX blocks.

18. Exporting Arbitration

Here are the rules for co-issuing exporting ALU clauses.

1) Position exports and position exports cannot be co-issued.
2) Position exports and memory exports cannot be co-issued.
3) Position exports and Z/Color exports cannot be co-issued.
4) Memory exports and Z/Color exports cannot be co-issued.
5) Memory exports and memory exports cannot be co-issued.
6) Z/color exports and Z/color exports cannot be co-issued.
7) Parameter exports and Z/Color exports CAN be co-issued.
8) Parameter exports and parameter exports CAN be co-issued.
9) Parameter exports and memory exports CAN be co-issued.

19. Real time commands

We are unable to use the parameter memory since there is no way for a command stream to write into it. Instead we need to add three 16x128 memories (one for each of three vertices x 16 interpolants). These will be mapped onto the register bus and written by type 0 packets, and output to the the parameter busses (the sequencer and/or PA need to be able to address the realtime parameter memory as well as the regular parameter store. For higher performance we should be able able to view them as two banks of 16 and do double buffering allowing one to be loaded, while the other is rasterized with. Most overlay shaders will need 2 or 4 scalar coordinates, one option might be to restrict the memory to 16x64 or 32x64 allowing only two interpolated scalars per cycle, the only problem I see with this is, if we view support for 16 vector-4 interpolants important (true only if we map ~~microsoft's~~Microsoft's high priority stream to the realtime stream), then the PA/sequencer need to support a realtime-specific mode where we need to address 32 vectors of parameters instead of 16.

20. State management

Formatted: Bullets and Numbering

Every clock, the sequencer will report to the CP the oldest states still in the pipe. These are the states of the programs as they enter the last ALU clause.

21. XY Address imports

Formatted: Bullets and Numbering

The SC will be able to send the XY addresses to the GPRs. It does so by interleaving the writes of the IJs (to the IJ buffer) with XY writes (to the XY buffer). Then when writing the data to the GPRs, the sequencer is going to interpolate the IJ data or pass the XY data thru a Fix→float converter and expander and write the converted values to the GPRs. The Xys are currently SCREEN SPACE COORDINATES. The values in the XY buffers will wrap.

~~20.~~22. Registers

Formatted: Bullets and Numbering

~~20.1~~22.1 Control

DYNAMIC_REG	Dynamic allocation (pixel/vertex) of the register file on or off.
VERTEX_REG_SIZE	What portion of the register file is reserved for vertices (static allocation only)
PIXEL_MIN_SIZE	Minimal size of the register file's pixel portion (dynamic only)
VERTEX_MIN_SIZE	Minimal size of the register file's vertex portion (dynamic only)
ARBITRATION_policy	policy of the arbitration between vertexes and pixels
CST_SIZE_P	Size of the constant store for pixels
CST_SIZE_V	Size of the constant store for vertexes
INST_STOR_ALLOC	interleaved, separate, interleaved+shared,separate+shared
VERTEX_WRAP	start point for the vertex instruction store (RT always ends at vertex_wrap and Begins at 0)
PIXEL_WRAP	start point for the pixel shader instruction store ~~(vertex shader always starts at 0)~~
~~SHAREDWRAP~~	~~start point for the shared instruction store~~
~~RTWRAP~~	~~start point for the RT instruction store (RT always ends at the end of the IM)~~
NO_INTERLEAVE	debug state register. Only allows one program at a time into the GPRs
NO_INTERLEAVE_ALU	debug state register. Only allows one ALU program at a time to be executed (instead of 2)
NO_PRED_OPTIMIZE	turns off the predicate bit optimization (conditional_execute_predicates is always executed.
INSTRUCTION_INDEX _PORT	This is where the CP puts the base address of the instruction writes and type (auto-incremented on reads/writes)
INSTRUCTION_DATA	This is where the CP puts the actual data going to the instruction memory
CONSTANT_DATA	This is where the CP puts constant data

~~20.2~~22.2 Context

Formatted: Bullets and Numbering

Vshader_fetch[7:0][7:0]	eight 8 bit pointers to the location where each clauses control program is located
Vshader_alu[7:0][7:0]	eight 8 bit pointers to the location where each clauses control program is located
Pshader_fetch[7:0][7:0]	eight 8 bit pointers to the location where each clauses control program is located
Pshader_alu[7:0][7:0]	eight 8 bit pointers to the location where each clauses control program is located
PSHADER	base pointer for the pixel shader
VSHADER	base pointer for the vertex shader
Vshader_cntl_size	size of the vertex shader (# of instructions in control program/2)
Pshader_cntl_size	size of the pixel shader (# of instructions in control program/2)
Pshader_size	size of the pixel shader (cntl+instructions)
Vshader_size	size of the vertex shader (cntl+instructions)
REG_ALLOC_PIX	number of registers to allocate for pixel shader programs
REG_ALLOC_VERT	number of registers to allocate for vertex shader programs
FLAT_GOUR[0…15]	which parameters are to be gouraud shaded
CYL_WRAP[0…63]	for which parameters (and channels (xyzw)) do we do the cyl wrapping.
P_export_mode	0xxxx : Normal mode

1xxxx : Multipass mode
If normal, bbbz where bbb is how many colors (0-4) and z is export z or not
If multipass 1-12 exports for color.
vshader_export_mask which of the last 6 ALU clauses is exporting
vshader_export_mode 0: position (1 vector), 1: position (2 vectors), 3:multipass
vshader_export_count[6] # of interpolated parameters exported in clause 7 OR
# of exported vectors to memory per clause in multipass mode (per clause)
Control_Flow 24 Dwords that contain the control flow constants.~~kill_vector_on use the mask kill vector to kill pixels and optimize texture pipe fetches OR use it as the fifth predicate vector which is the only predicate vector kept across clause boundaries.~~

~~21.~~23. DEBUG registers

Formatted: Bullets and Numbering

~~21.1 Control~~

~~Shader_pipe # of the shader pipe for method 3 (0…64)~~
~~Count_+clause instruction count and clause number for method 3~~
~~Dword_select channel select for method 3~~
~~Mode operating mode for method 3~~
~~Rstate render state method 3 is operating on~~
~~Vector_count vector number the method 3 will export~~

~~21.2~~23.1 Context

Formatted: Bullets and Numbering

PROB_ADDR instruction address where the first problem occurred
PROB_COUNT number of problems encountered during the execution of the program
Count instruction counter for debug method 2
Addr break address for method number 2
Clause_mode[3] clause mode for debug method 2

~~22.~~24. Interfaces

Formatted: Bullets and Numbering

~~22.1~~24.1 External Interfaces

Whenever an x is used, it means that the bus is broadcast to all units of the same name. For example, if a bus is named SQ→SPx it means that SQ is going to broadcast the same information to all SP instances.

~~22.1.1~~24.1.1 *~~PA/~~SC to SP~~0~~ : IJ bus*

Formatted: Bullets and Numbering

This is a bus that sends the IJ information to the IJ fifos on the top of each shader pipe. At the same time the control information goes to the sequencer. There are 4 of these buses over the whole chip (SP0 thru 3)

Name	Direction	Bits	Description
~~IJs~~SC_SP0_data	~~PA~~SC→SP0	64	IJ information sent over 2 clocks (or XY info sent over 1 clock in the lower 24 LSBs of the interface)
~~Mask~~SC_SP0_q_wr_mask	~~PA~~SC→SP0	1	Write Mask
SC_SP0_dest	SC→SP0	1	Controls the write destination (XY buffer, IJ buffer)
SC_SP1_data	SC→SP1	64	IJ information sent over 2 clocks (or XY info sent over 1 clock in the lower 24 LSBs of the interface)
SC_SP1_q_wr_mask	SC→SP1	1	Write Mask
SC_SP1_dest	SC→SP1	1	Controls the write destination (XY buffer, IJ buffer)
SC_SP2_data	SC→SP2	64	IJ information sent over 2 clocks (or XY info sent over 1 clock in the lower 24 LSBs of the interface)
SC_SP2_q_wr_mask	SC→SP2	1	Write Mask
SC_SP2_dest	SC→SP2	1	Controls the write destination (XY buffer, IJ buffer)
SC_SP3_data	SC→SP3	64	IJ information sent over 2 clocks (or XY info sent over 1 clock in the lower 24 LSBs of the interface)
SC_SP3_q_wr_mask	SC→SP3	1	Write Mask
SC_SP3_dest	SC→SP3	1	Controls the write destination (XY buffer, IJ buffer)

~~22.1.2~~24.1.2 ~~PA/~~SC to SEQ : IJ Control bus

Formatted: Bullets and Numbering

This is the control information sent to the sequencer in order to control the IJ fifos and all other information needed to execute a shader program on the sent pixels. This information is sent over 2 clocks, if SENDXY is asserted the next control packet is going to be ignored and XY information is going to be sent on the IJ bus (for the quads that where just sent).

Name	Direction	Bits	Description
~~Write Mask~~SC_SQ_q_wr_mask	~~PA~~SC→S~~E~~Q~~(SP)~~	4	Quad Write mask left to right
~~LOD_CORRECT~~SC_SQ_lod_correct	SC→SQ~~PA→SEQ(SP)~~	24	LOD correction per quad (6 bits per quad)
~~FVTX~~SC_SQ_flat_vertex	SC→SQ~~PA→SEQ(SP)~~	2	Provoking vertex for flat shading
~~PPTR0~~SC_SQ_param_ptr0	SC→SQ~~PA→SEQ(SP)~~	11	P Store pointer for vertex 0
SC_SQ_param_ptr1~~PPRT1~~	SC→SQ~~PA→SEQ(SP)~~	11	P Store pointer for vertex 1
SC_SQ_param_ptr2~~PPTR2~~	SC→SQ~~PA→SEQ(SP)~~	11	P Store pointer for vertex 2
SC~~E_OFF_VECTOR~~_SQ_end_of_vect	SC→SQ~~PA→SEQ(SP)~~	1	End of the vector
~~DEALLOC~~SC_SQ_store_dealloc	SC→SQ~~PA→SEQ(SP)~~	1	Deallocation token for the P Store
~~STATE~~SC_SQ_state	SC→SQ~~PA→SEQ(SP)~~	~~21~~3	State/constant pointer (6*3+3)
~~VALID~~SC_SQ_valid_pixel	SC→SQ~~PA→SEQ(SP)~~	16	Valid bits for all pixels
~~NULL~~SC_SQ_null_prim	SC→SQ~~PA→SEQ(SP)~~	1	Null Primitive (for PC deallocation purposes)
SC_SQ_end_of_prim~~E_OFF_PRIM~~	SC→SQ~~PA→SEQ(SP)~~	1	End Of the primitive
~~FBFACE~~SC_SQ_fbface	SC→SQ~~PA→SEQ(SP)~~	1	Front face = 1, back face = 0
SC_SQ_send_xy	SC→SQ	1	Sending XY information [XY information is going to be sent on the next clock]
~~TYPE~~SC_SQ_prim_type	SC→SQ~~PA→SEQ(SP)~~	3	Stippled line and Real time command need to load tex cords from alternate buffer 000 : Normal ~~001 : Stippled line~~ 011 : Real Time 100 : Line AA 101 : Point AA 110 : Sprite
SC_SQ_RTRn	~~SEQ→PA~~SQ→SC	1	Stalls the PA in n clocks
SC_SQ_RTS	~~PA~~SC→~~SEQ(SP)~~SQ	1	~~PA~~SC ready to send data

~~22.1.3~~24.1.3 S~~E~~Q to SP~~0 ~~: Interpolator bus

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
~~TYPE~~SQ_SPx_interp_prim_type	S~~E~~Q→SPx~~0~~	3	Type of the primitive 000 : Normal ~~001 : Stippled line/Poly~~ 011 : Real Time 100 : Line AA 101 : Point AA 110 : Sprite
~~FVTX~~SQ_SPx_interp_flat_vtx	S~~E~~Q→SP~~0~~x	2	Provoking vertex for flat shading

~~FLAT_GOURAUD~~SQ_SPx_interp_flat_gouraud	S~~E~~Q→SP~~0~~x	1	Flat or gouraud shading
SQ_SPx_interp_cyl_wrap~~CYL_WRAP~~	S~~E~~Q→SP~~0~~x	4	Wich parameter needs to be cylindrical wrapped
SQ_SPx_interp_ijline~~IJ_Line number~~	S~~E~~Q→SPx~~0~~	2	Line in the IJ/XY buffer to use to interpolate
SQ_SPx_interp_buff_swap~~Swap_Buffers~~	S~~E~~Q→SP~~0~~x	1	Swap the IJ/XY buffers at the end of the interpolation
SQ_SPx_interp_ij_xy	SQ→SPx	1	Read from the IJ buffer or from the XY buffer
SQ_SPx_interp_param0~~Param_0~~	S~~E~~Q→SP~~0~~x	1	We are interpolating parameter 0

~~22.1.4~~24.1.4 S~~E~~Q to SP~~0~~ : Parameter ~~-~~Cache Read control bus

Formatted: Bullets and Numbering

The four following interfaces (SQ→SP, SQ→SX,SP→SX and SX→Interpolators) are all SYNCHRONIZED together.

Name	Direction	Bits	Description
SQ_SPx_~~P~~ptr0~~1~~	S~~E~~Q→SPx~~0~~	~~7~~9	Pointer of PC ~~(7 LSBs of Pointer)~~
SQ_SPx_~~P~~ptr1~~2~~	S~~E~~Q→SPx~~P0~~	~~7~~9	Pointer of PC ~~(7 LSBs of Pointer)~~
SQ_SPx_~~P~~ptr~~3~~2	S~~E~~Q→SPx~~0~~	9~~7~~	Pointer of PC ~~(7 LSBs of Pointer)~~
SQ_SP0_read_ena	SQ→SP0	4	Read enables for the 4 memories in the SP0
SQ_SP1_read_ena	SQ→SP1	4	Read enables for the 4 memories in the SP1
SQ_SP2_read_ena	SQ→SP2	4	Read enables for the 4 memories in the SP2
SQ_SP3_read_ena	SQ→SP3	4	Read enables for the 4 memories in the SP3

~~22.1.5~~24.1.5 S~~E~~Q to SX~~0~~ : Parameter Cache Mux control Bus

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SXx_~~M~~mux0~~1~~	S~~E~~Q→SXx~~0~~	4	Mux control for PC (4 MSbs of Pointer)
SQ_SXx_~~M~~mux~~2~~1	S~~E~~Q→SXx~~0~~	4	Mux control for PC (4 MSbs of Pointer)
SQ_SXx_~~M~~mux~~3~~2	S~~E~~Q→SXx~~0~~	4	Mux control for PC (4 MSbs of Pointer)

24.1.6 SP to SX: Parameter data

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SP0_SX0_data0	SP0→SX0	128	Parameter data 0
SP0_SX0_data1	SP0→SX0	128	Parameter data 1
SP0_SX0_data2	SP0→SX0	128	Parameter data 2
SP0_SX0_data3	SP0→SX0	128	Parameter data 3
SP1_SX1_data0	SP1→SX1	128	Parameter data 0
SP1_SX1_data1	SP1→SX1	128	Parameter data 1
SP1_SX1_data2	SP1→SX1	128	Parameter data 2
SP1_SX1_data3	SP1→SX1	128	Parameter data 3
SP2_SX0_data0	SP2→SX0	128	Parameter data 0
SP2_SX0_data1	SP2→SX0	128	Parameter data 1
SP2_SX0_data2	SP2→SX0	128	Parameter data 2
SP2_SX0_data3	SP2→SX0	128	Parameter data 3
SP3_SX1_data0	SP3→SX1	128	Parameter data 0
SP3_SX1_data1	SP3→SX1	128	Parameter data 1
SP3_SX1_data2	SP3→SX1	128	Parameter data 2
SP3_SX1_data3	SP3→SX1	128	Parameter data 3

~~22.1.6~~24.1.7 SX~~0~~ to ~~SP0~~ Interpolators: Parameter Cache Return bus

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SXx_SPx_~~V~~vtx_data_~~1~~0	SXx~~0~~→SPx~~0~~	128	Vertex data to interpolate
SXx_SPx_~~V~~vtx_data_~~2~~1	SXx~~0~~→SPx~~0~~	128	Vertex data to interpolate
SXx_SPx_~~V~~vtx_data_~~3~~2	SXx~~0~~→SPx~~0~~	128	Vertex data to interpolate

~~22.1.7~~24.1.8 VGT to SP~~0~~0/S~~E~~Q : Vertex Bus

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
VGT_SP0_vrtx_indexes~~Vertex indexes~~	VGT→SP0	128	Pointers of indexes or HOS surface information
VGT_SP0_end_of_vect~~EOF_vector~~	VGT→SP0~~/SEQ~~	1	End of the vector
VGT_SP0_vrtx_format~~Inputs_vert~~	VGT→SP0~~/SEQ~~	1	0: Normal 128 bits per vert 1: double 256 bits per vert
VGT_SQ_end_of_vect	VGT→SQ	1	End of the vector
VGT_SQ_vrtx_format	VGT→SQ	1	0: Normal 128 bits per vert 1: double 256 bits per vert
VGT_SQ_state~~STATE~~	VGT→S~~E~~Q	~~21~~3	Render State (6*3+3 for constants)

~~22.1.8 CP to SEQ : Constant store load~~

Formatted: Bullets and Numbering

~~22.1.9CP to SEQ : Fetch State store load~~

~~22.1.10CP to SEQ : Control State store load~~

Formatted: Bullets and Numbering

~~{ISSUE: How,Who and what is the size of this bus?}~~

~~22.1.11 MH to SEQ: Instruction store Load~~

Formatted: Bullets and Numbering

24.1.9 SEQ to CP: State report

Name	Direction	Bits	Description
SQ_CP_vrtx_ state	SEQ→CP	3	Oldest vertex state still in the pipe
SQ_CP_pix_state	SEQ→CP	3	Oldest pixel state still in the pipe

~~{ISSUE: CP or MH?}~~

~~22.1.12~~24.1.10 SP~~0~~ to SX~~0~~ : Pixel/Vertex ~~read from RBs~~write to SX

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SP0_SX0_Export_data	SP0→SX0	~~64*16~~256	4~~32~~ pair~~s~~s of 32 bits channel values
SP0_SX0_Shader_Dest	SP0→SX0	4	Specifies one of the of up to 12 export destinations
SP1_SX1_Export_data	SP1→SX1	256	4 pairs of 32 bits channel values
SP1_SX1_Shader_Dest	SP1→SX1	4	Specifies one of the of up to 12 export destinations
SP2_SX0_Export_data	SP2→SX0	256	4 pairs of 32 bits channel values
SP2_SX0_Shader_Dest	SP2→SX0	4	Specifies one of the of up to 12 export destinations
SP3_SX1_Export_data	SP3→SX1	256	4 pairs of 32 bits channel values
SP3_SX1_Shader_Dest	SP3→SX1	4	Specifies one of the of up to 12 export destinations
SPx_SXx_Shader_Count	SP0→SX0	3	Each set of four pixels or vectors is exported over eight clocks. This field specifies where the SP is in that sequence.
SPx_SXx_Shader_Last	SP0→SX0	1	~~The current export clause is over (true for one clock)~~ ~~The last export instruction creates *two* cycles to the RB. This needs to be set on or after the last RB cycle that is produced by the last export instruction, but before the first RB cycle of the first export instruction of the next clause.~~Asserted on the first shader count of the last export of the clause
SP0_SX0_Shader_PixelValid	SP0→SX0	4~~x4~~	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock

SP0_SX0_Shader_WordValid	SP0→SX0	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors
SP1_SX1_Shader_PixelValid	SP1→SX1	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SP1_SX1_Shader_WordValid	SP1→SX1	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors
SP2_SX0_Shader_PixelValid	SP2→SX0	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SP2_SX0_Shader_WordValid	SP2→SX0	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors
SP3_SX1_Shader_PixelValid	SP3→SX1	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SP3_SX1_Shader_WordValid	SP3→SX1	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors

~~22.1.13~~24.1.11 *S~~E~~Q to SX~~X0~~ : Control bus*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SXx_Export_Pixel	S~~E~~Q→SXx~~0~~	1	1: Pixel 0: Vertex
SQ_SXx_Export_SEND	S~~E~~Q→SXx~~0~~	1	Raised to indicate that the SQ is starting an export
SQ_SXx_Export_Clause	S~~E~~Q→SXx~~0~~	3	Clause number, which is needed for vertex clauses
SQ_SXx_Export_State	S~~E~~Q→SXx~~0~~	~~21?~~3	State ID, which is needed for vertex clauses

These fields are sent synchronously with SP export data, described in SP0→SX0 interface
{ISSUE: Where are the PC pointers}

~~22.1.14~~24.1.12 *SX~~0~~ to S~~E~~Q : Output file control*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SXx_SQ_Export_~~RTS~~count_rdy	~~SX0~~SXx→S~~E~~Q	1	Raised by SX0 to indicate that the following two fields reflect the result of the most recent export
SXx_SQ_Export_Position	SXx~~0~~→S~~E~~Q	1	Specifies whether there is room for another position.
SXx_SQ_Export_Buffer	SX~~0~~x→S~~E~~Q	7	Specifies the space ~~availble~~available in the output buffers. 0: buffers are full 1: 2K-bits available (32-bits for each of the 64 pixels in a clause) ... 64: 128K-bits available (16 128-bit entries for each of 64 pixels) 65-127: RESERVED

~~22.1.15 SP0 to SX0 : Position return bus~~

Formatted: Bullets and Numbering

~~22.1.16~~24.1.13 Shader Engine to Fetch Unit Bus ~~(Fast Bus)~~

Four quad's worth of addresses is transferred to Fetch Unit every clock. These are sourced from a different pixel within each of the sub-engines repeating every 4 clocks. The register file index to read must precede the data by 2 clocks. The Read address associated with Quad 0 must be sent 1 clock after the Instruction Start signal is sent, so that data is read 3 clocks after the Instruction Start.

Four Quad's worth of Fetch Data may be written to the Register file every clock. These are directed to a different pixel of the sub-engines repeating every 4 clocks. The register file index to write must accompany the data. Data and Index associated with the Quad 0 must be sent 3 clocks after the Instruction Start signal is sent.

Name	Direction	Bits	Description
~~Tex_RegFile_Read_Data~~SP0_TP0_fetch_addr	SP0->T~~EX~~P0	~~2048~~512	~~16~~ 4 Fetch Addresses read from the Register file
~~Tex_RegFile_Write _Data~~TP0_SP0_data	TP0~~EX~~→SP0	~~2048~~512	~~16~~ 4 texture results
SP1_TP1_fetch_addr	SP1->TP1	512	4 Fetch Addresses read from the Register file
TP1_SP1_data	TP1→SP1	512	4 texture results
SP2_TP2_fetch_addr	SP2->TP2	512	4 Fetch Addresses read from the Register file
TP2_SP2_data	TP2→SP2	512	4 texture results
SP3_TP3_fetch_addr	SP3->TP3	512	4 Fetch Addresses read from the Register file
TP3_SP3_data	TP3→SP3	512	4 texture results
TPx_SPx_gpr_dst	TPx→SPx	7	Write address into the gprs
TPx_SPx_gpr_cmask	TPx→SPx	4	Channel mask

~~22.1.17~~24.1.14 Sequencer to Fetch Unit bus ~~(Slow Bus)~~

Formatted: Bullets and Numbering

Once every ~~four~~ clock, the fetch unit sends to the sequencer on ~~wich~~which clause it is now working and if the data in the registers is ready or not. This way the sequencer can update the fetch counters for the reservation station fifos. The sequencer also provides the ~~intruction~~instruction and constants for the fetch to execute and the address in the register file where to write the fetch return data.

Name	Direction	Bits	Description
~~Tex_Ready~~TPx_SQ_data_rdy	T~~EX~~Px→ S~~E~~Q	1	Data ready
TPx_SQ_clause_num~~Tex_Clause_Num~~	T~~EX~~Px→ S~~E~~Q	3	Clause number
SQ_TPx_const~~Tex_cst~~	S~~E~~Q→T~~EX~~Px	~~10~~64	Fetch state ~~address 10 bits~~ sent over 4 clocks
~~Tex_Inst~~SQ_TPx_instuct	S~~E~~Q→T~~EX~~Px	~~12~~24	Fetch instruction ~~address 12 bits~~ sent over 4 clocks
SQ_TPx_end_of_clause~~EO_CLAUSE~~	S~~E~~Q→T~~EX~~Px	1	Last instruction of the clause
~~PHASE~~SQ_TPx_phase	S~~E~~Q→T~~EX~~Px	~~1~~2	Write phase signal
SQ_TP0_lod_correct~~LOD CORRECT~~	S~~E~~Q→T~~EX~~P0	6~~96~~	LOD correct 3 bits per comp 2 components per quad ~~* 16~~ quads
~~Mask~~SQ_TP0_pmask	S~~E~~Q→T~~EX~~P0	~~64~~4	Pixel mask 1 bit per pixel
SQ_TP1_lod_correct	SQ→TP1	6	LOD correct 3 bits per comp 2 components per quad quads
SQ_TP1_pmask	SQ→TP1	4	Pixel mask 1 bit per pixel
SQ_TP2_lod_correct	SQ→TP2	6	LOD correct 3 bits per comp 2 components per quad quads
SQ_TP2_pmask	SQ→TP2	4	Pixel mask 1 bit per pixel
SQ_TP3_lod_correct	SQ→TP3	6	LOD correct 3 bits per comp 2 components per quad quads
SQ_TP3_pmask	SQ→TP3	4	Pixel mask 1 bit per pixel
~~Tex_Clause_Num~~SQ_TPx_clause_num	S~~E~~Q→T~~EX~~Px	3	Clause number
~~Tex_Write_Register_Index~~SQ_TPx_write_gpr_index	S~~E~~Q->T~~EX~~Px	7	Index into Register file for write of

			returned Fetch Data

24.1.15 Sequencer to SP: GPR control

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SPx_gpr_wr_addr	SQ→SPx	7	Write address
SQ_SPx_gpr_rd_addr	SQ→SPx	7	Read address
SQ_SPx_gpr_re_addr	SQ→SPx	1	Read Enable
SQ_SPx_gpr_we_addr	SQ→SPx	1	Write Enable
SQ_SPx_gpr_phase_mux	SQ→SPx	2	The phase mux
SQ_SPx_gpr_channel_mask	SQ→SPx	4	The channel mask
SQ_SP0_gpr_pixel_mask	SQ→SP0	4	The pixel mask
SQ_SP1_gpr_pixel_mask	SQ→SP1	4	The pixel mask
SQ_SP2_gpr_pixel_mask	SQ→SP2	4	The pixel mask
SQ_SP3_gpr_pixel_mask	SQ→SP3	4	The pixel mask

24.1.16 Sequencer to SPx: Parameter cache write control

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SPx_pc_wr_addr	SQ→SPx	7	Write address
SQ_SPx_pc_we_addr	SQ→SPx	1	Write Enable
SQ_SPx_pc_phase_mux	SQ→SPx	1	The output selector mux (gpr vs parameter cache)

24.1.17 Sequencer to SPx: Instructions

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SPx_instruct_start	SQ→SPx	1	Instruction start
SQ_SP_instruct	SQ→SPx	20	Instruction sent over 4 clocks
SQ_SPx_stall	SQ→SPx	1	Stall signal
SQ_SPx_Shader_Count	SQ→SPx	3	Each set of four pixels or vectors is exported over eight clocks. This field specifies where the SP is in that sequence.
SQ_SPx_Shader_Last	SQ→SPx	1	Asserted on the first shader count of the last export of the clause
SQ_SP0_Shader_PixelValid	SQ→SP0	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP0_Shader_WordValid	SQ→SP0	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors
SQ_SP1_Shader_PixelValid	SQ→SP1	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP1_Shader_WordValid	SQ→SP1	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors
SQ_SP2_Shader_PixelValid	SQ→SP2	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP2_Shader_WordValid	SQ→SP2	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors
SQ_SP3_Shader_PixelValid	SQ→SP3	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP3_Shader_WordValid	SQ→SP3	2	Specifies whether to write low and/or high 32-bit

			word of the 64-bit export data from each of the 16 pixels or vectors

24.1.18 SP to Sequencer: Constant address load

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SP0_SQ_const_addr	SP0→SQ	36	Constant address load to the sequencer
SP0_SQ_valid	SP0→SQ	1	Data valid
SP1_SQ_const_addr	SP1→SQ	36	Constant address load to the sequencer
SP1_SQ_valid	SP1→SQ	1	Data valid
SP2_SQ_const_addr	SP2→SQ	36	Constant address load to the sequencer
SP2_SQ_valid	SP2→SQ	1	Data valid
SP3_SQ_const_addr	SP3→SQ	36	Constant address load to the sequencer
SP3_SQ_valid	SP3→SQ	1	Data valid

Formatted

Formatted

Formatted

Formatted

24.1.19 Sequencer to SPx: constant broadcast

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SPx_constant	SQ→SPx	128	Constant broadcast

Formatted

Formatted

Formatted

Formatted

24.1.20 SP0 to Sequencer: Kill vector load

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SP0_SQ_kill_vect	SP0→SQ	4	Kill vector load
SP1_SQ_kill_vect	SP1→SQ	4	Kill vector load
SP2_SQ_kill_vect	SP2→SQ	4	Kill vector load
SP3_SQ_kill_vect	SP3→SQ	4	Kill vector load

24.1.21 SQ to CP: RBBM bus

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_RBB_rs	SQ→CP	1	Read Strobe
SQ_RBB_rd	SQ→CP	32	Read Data
SQ_RBBM_nrtrtr	SQ→CP	1	Optional
SQ_RBBM_rtr	SQ→CP	1	Real-Time (Optional)

24.1.22 CP to SQ: RBBM bus

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
rbbm_we	CP→SQ	1	Write Enable
rbbm_a	CP→SQ	18	Address -- Upper Extent is TBD
rbbm_wd	CP→SQ	32	Data
rbbm_be	CP→SQ	4	Byte Enables
rbbm_re	CP→SQ	1	Read Enable
rbb_rs0	CP→SQ	1	Read Return Strobe 0
rbb_rs1	CP→SQ	1	Read Return Strobe 1
rbb_rd0	CP→SQ	32	Read Data 0
rbb_rd1	CP→SQ	32	Read Data 0
RBBM_SQ_soft_reset	CP→SQ	1	Soft Reset

~~23.~~25. Examples of program executions

Formatted: Bullets and Numbering

~~23.1.1~~25.1.1 Sequencer Control of a Vector of Vertices

1. PA sends a vector of 64 vertices (actually vertex indices – 32 bits/index for 2048 bit total) to the RE's Vertex FIFO
   - state pointer as well as tag into position cache is sent along with vertices
   - space was allocated in the position cache for transformed position before the vector was sent

- **also before the vector is sent to the RE, the CP has loaded the global instruction store with the vertex shader program (using the MH?)**
- The vertex program is assumed to be loaded when we receive the vertex vector.
  - the SEQ then accesses the IS base for this shader using the local state pointer (provided to all sequencers by the RBBM when the CP is done loading the program)

2. SEQ arbitrates between the Pixel FIFO and the Vertex FIFO – basically the Vertex FIFO always has priority
   - at this point the vector is removed from the Vertex FIFO
   - the ~~arbitrer~~arbiter is not going to select a vector to be transformed if the parameter cache is full unless the pipe as nothing else to do (ie no pixels are in the pixel fifo).

3. SEQ allocates space in the SP register file for index data plus GPRs used by the program
   - the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer that came down with the vertices
   - SEQ will not send vertex data until space in the register file has been allocated

4. SEQ sends the vector to the SP register file over the RE_SP interface (which has a bandwidth of 2048 bits/cycle)
   - the 64 vertex indices are sent to the 64 register files over 4 cycles
     - RF0 of SU0, SU1, SU2, and SU3 is written the first cycle
     - RF1 of SU0, SU1, SU2, and SU3 is written the second cycle
     - RF2 of SU0, SU1, SU2, and SU3 is written the third cycle
     - RF3 of SU0, SU1, SU2, and SU3 is written the fourth cycle
   - the index is written to the least significant 32 bits **(floating point format?) (what about compound indices)** of the 128-bit location within the register file (w); the remaining data bits are set to zero (x, y, z)

5. SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of fetch state machine 0, or TSM0 FIFO)
   - the control packet contains the state pointer, the tag to the position cache and a register file base pointer.

6. TSM0 accepts the control packet and fetches the instructions for fetch clause 0 from the global instruction store
   - TSM0 was first selected by the TSM arbiter before it could start

7. all instructions of fetch clause 0 are issued by TSM0

8. the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)
   - TSM0 does not wait for requests made to the Fetch Unit to complete; it passes the register file write index for the fetch data to the TU, which will write the data to the RF as it is received
   - once the TU has written all the data to the register files, it increments a counter that is associated with ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go ahead start to execute the ALU clause

9. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store

10. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of fetch state machine 1, or TSM1 FIFO)

11. the control packet continues to travel down the path of reservation stations until all clauses have been executed
    - position can be exported in ALU clause 3 (or 4?); the data (and the tag) is sent over a position bus (which is shared with all four shader pipes) back to the PA's position cache
    - A parameter cache pointer is also sent along with the position data. This tells to the PA where the data is going to be in the parameter cache.
      - there is a position export FIFO in the SP that buffers position data before it gets sent back to the PA
      - the ASM arbiter will prevent a packet from starting an exporting clause if the position export FIFO is full
    - parameter data is exported in clause 7 (as well as position data if it was not exported earlier)
      - parameter data is sent to the Parameter Cache over a dedicated bus
      - the SEQ allocates storage in the Parameter Cache, and the SEQ deallocates that space when there is no longer a need for the parameters (it is told by the PA when using a token).
      - the ASM arbiter will prevent a packet from starting on ASM7 if the parameter cache (or the position buffer if position is being exported) is full

12. after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program

~~23.1.2~~25.1.2 *Sequencer Control of a Vector of Pixels*

Formatted: Bullets and Numbering

1. **As with vertex shader programs, pixel shaders are loaded into the global instruction store by the CP**
   - At this point it is assumed that the pixel program is loaded into the instruction store and thus ready to be read.
2. the RE's Pixel FIFO is loaded with the barycentric coordinates for pixel quads by the detailed walker
   - the state pointer and the LOD correction bits are also placed in the Pixel FIF0
   - the Pixel FIFO is wide enough to source four quad's worth of barycentrics per cycle
3. SEQ arbitrates between Pixel FIFO and Vertex FIFO – when there are no vertices pending OR there is no space left in the register files for vertices, the Pixel FIFO is selected
4. SEQ allocates space in the SP register file for all the GPRs used by the program
   - the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer
   - SEQ will not allow interpolated data to be sent to the shader until space in the register file has been allocated
5. SEQ controls the transfer of interpolated data to the SP register file over the RE_SP interface (which has a bandwidth of 2048 bits/cycle). See interpolated data bus diagrams for details.
6. SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of fetch state machine 0, or TSM0 FIFO)
   - note that there is a separate set of reservation stations/arbiters/state machines for vertices and for pixels
   - the control packet contains the state pointer, the register file base pointer, and the LOD correction bits
   - all other ~~informations~~information (such as quad address for example) travels in a separate FIFO
7. TSM0 accepts the control packet and fetches the instructions for fetch clause 0 from the global instruction store
   - TSM0 was first selected by the TSM arbiter before it could start
8. all instructions of fetch clause 0 are issued by TSM0
9. the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)
   - TSM0 does not wait for fetch requests made to the Fetch Unit to complete; it passes the register file write index for the fetch data to the TU, which will write the data to the RF as it is received
   - once the TU has written all the data for a particular clause to the register files, it increments a counter that is associated with the ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go ahead and pop the FIFO and start to execute the ALU clause
10. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store
11. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of fetch state machine 1, or TSM1 FIFO)
12. the control packet continues to travel down the path of reservation stations until all clauses have been executed
   - pixel data is exported in the last ALU clause (clause 7)
      - it is sent to an output FIFO where it will be picked up by the render backend
      - the ASM arbiter will prevent a packet from starting on ASM7 if the output FIFO is full
13. after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program

~~23.1.3~~25.1.3 *Notes*

Formatted: Bullets and Numbering

14. The state machines and ~~arbitrers~~arbiters will operate ahead of time so that they will be able to immediately start the real threads or stall.

15. The register file base pointer for a vector needs to travel with the vector through the reservation stations, but the instruction store base pointer does not – this is because the RF pointer is different for all threads, but the IS pointer is only different for each state and thus can be accessed via the state pointer

16. Waterfalling still needs to be specked out.

~~24.~~26. Open issues

Formatted: Bullets and Numbering

There is currently an issue with constants. If the constants are not the same for the whole vector of vertices, we don't have the ~~bandwith~~bandwidth from the fetch store to feed the ALUs. Two solutions exists for this problem:

1) Let the compiler handle the case and put those instructions in a fetch clause so we can use the ~~bandwith~~bandwidth there to operate. This requires a significant amount of temporary storage in the register store.
2) Waterfall down the pipe allowing only at a given time the vertices having the same constants to operate in ~~parrallel~~parallel. This might in the worst case slow us down by a factor of 16.

Need to do some testing on the size of the register file as well as on the register file allocation method (dynamic VS static).

Saving power?

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 2015~~5~~ ~~December, 200126~~	GEN-CXXXXX-REVA	1 of 43

Author: Laurent Lefebvre

Issue To:

Copy No:

R400 Sequencer Specification

S~~E~~Q

Version 1.4~~3~~

Overview: This is an architectural specification for the R400 Sequencer block (SEQ). It provides an overview of the required capabilities and expected uses of the block. It also describes the block interfaces, internal sub-blocks, and provides internal state diagrams.

AUTOMATICALLY UPDATED FIELDS:
Document Location:
C:\perforce\r400\doc_lib\design\blocks\sq\R400_Sequencer.doc~~C:\perforce\r400\doc_lib\parts\sq\R400_Sequencer.docC:\perforce\r400\arch\doc\gfx\RE\R400_Sequencer.doc~~

APPROVALS	
Name/Dept	Signature/Date

Remarks:

THIS DOCUMENT CONTAINS CONFIDENTIAL INFORMATION THAT COULD BE SUBSTANTIALLY DETRIMENTAL TO THE INTEREST OF ATI TECHNOLOGIES INC. THROUGH UNAUTHORIZED USE OR DISCLOSURE.

ATI 2020
LG v. ATI
IPR2015-00326

Table Of Contents

1. OVERVIEW .... 6~~4~~
1.1 Top Level Block Diagram .... 8~~6~~
1.2 Data Flow graph .... 10~~8~~
1.3 Control Graph .... 11~~9~~
2. INTERPOLATED DATA BUS .... 11~~9~~
3. INSTRUCTION STORE .... 14~~12~~
4. SEQUENCER INSTRUCTIONS .... 16~~14~~
5. CONSTANT STORES .... 16~~14~~
5.1 Memory organizations .... 16~~14~~
5.2 Management of the remaping tables .... 16~~14~~
5.3 Constant Store Indexing .... 16~~14~~
5.4 .... 17~~14~~
5.5 Real Time Commands .... 17~~15~~
6. LOOPING AND BRANCHES .... 17~~15~~
6.1 The controlling state .... 17~~15~~
6.2 The Control Flow Program .... 18~~15~~
6.3 Data dependant predicate instructions .... 21~~18~~
6.4 HW Detection of PV,PS .... 21~~18~~
6.5 Register file indexing .... 21~~18~~
6.6 Predicated Instruction support for Texture clauses .... 22~~19~~
6.7 Debugging the Shaders .... 22~~19~~
6.7.1 Method 1: Debugging registers .... 22~~19~~
6.7.2 Method 2: Exporting the values in the GPRs (12) 22~~20~~
7. PIXEL KILL MASK .... 23~~20~~
8. MULTIPASS VERTEX SHADERS (HOS) .... 23~~20~~
9. REGISTER FILE ALLOCATION .... 23~~20~~
10. FETCH ARBITRATION .... 24~~21~~
11. ALU ARBITRATION .... 24~~21~~
12. HANDLING STALLS .... 25~~22~~
13. CONTENT OF THE RESERVATION STATION FIFOS 25~~22~~
14. THE OUTPUT FILE .... 25~~22~~
15. IJ FORMAT .... 25~~22~~
15.1 Interpolation of constant attributes .... 26~~23~~
16. THE PARAMETER CACHE .... 26~~23~~
17. VERTEX POSITION EXPORTING .... 26~~23~~
18. EXPORTING ARBITRATION .... 26~~23~~
19. EXPORT TYPES .... 27~~24~~
19.1 Vertex Shading .... 27~~24~~
19.2 Pixel Shading .... 27~~24~~
20. SPECIAL INTERPOLATION MODES .... 27~~24~~
20.1 Real time commands .... 27~~24~~
20.2 Sprites/ XY screen coordinates/ FB information .... 28~~25~~
20.3 Auto generated counters .... 28~~25~~
20.3.1 Vertex shaders .... 28~~25~~
20.3.2 Pixel shaders .... 28~~25~~
21. STATE MANAGEMENT .... 29~~26~~
22. XY ADDRESS IMPORTS .... 29~~26~~
22.1 Vertex indexes imports .... 29~~26~~
23. REGISTERS .... 29~~26~~
23.1 Control .... 29~~26~~
23.2 Context .... 30~~27~~
24. DEBUG REGISTERS .... 31~~28~~
24.1 Context .... 31~~28~~
25. INTERFACES .... 31~~28~~
25.1 External Interfaces .... 31~~28~~
25.1.1 SC to SP : IJ bus .... 31~~28~~
25.1.2 SC to SQ : IJ Control bus .... 32~~28~~
25.1.3 SQ to SC: Vertex/Pixel shader synchronization bus .... **Error! Bookmark not defined.**~~29~~
25.1.4 SQ to SP: Interpolator bus .... 32~~29~~
25.1.5 SQ to SP: GPR Input Mux select .... 33~~29~~
25.1.6 SQ to SPx: Parameter cache write control 33~~30~~
25.1.7 SQ to SP: Parameter Cache Read control bus 33~~30~~
25.1.8 SQ to SX: Parameter Cache Mux control Bus 33~~30~~
25.1.9 SP to SX: Parameter data .... 33~~30~~
25.1.10 SX to Interpolators: Parameter Cache Return bus 34~~30~~
25.1.11 SQ to SP0: Staging Register Data 34~~31~~
25.1.12 PA to SQ : Vertex interface .... 34~~31~~
25.1.13 SQ to CP: State report .... 37~~34~~
25.1.14 SP to SX : Pixel/Vertex write to SX 37~~34~~
25.1.15 SQ to SX: Control bus .... 37~~34~~
25.1.16 SX to SQ : Output file control .... 38~~35~~
25.1.17 Shader Engine to Fetch Unit Bus .... 38~~35~~
25.1.18 Sequencer to Fetch Unit bus .... 38~~35~~
25.1.19 Sequencer to SP: GPR control .... 39~~36~~
25.1.20 Sequencer to SPx: Instructions .... 39~~36~~
25.1.21 SP to Sequencer: Constant address load 40~~37~~
25.1.22 Sequencer to SPx: constant broadcast 40~~37~~
25.1.23 SP0 to Sequencer: Kill vector load 40~~37~~
25.1.24 SQ to CP: RBBM bus .... 40~~37~~
25.1.25 CP to SQ: RBBM bus .... 40~~37~~
26. EXAMPLES OF PROGRAM EXECUTIONS 41~~37~~
26.1.1 Sequencer Control of a Vector of Vertices 41~~37~~
26.1.2 Sequencer Control of a Vector of Pixels 42~~38~~
26.1.3 Notes .... 43~~39~~
27. OPEN ISSUES .... 43~~40~~
~~**1. OVERVIEW .... 4**~~
~~1.1 Top Level Block Diagram .... 6~~
~~1.2 Data Flow graph .... 8~~
~~1.3 Control Graph .... 9~~
~~**2. INTERPOLATED DATA BUS .... 9**~~

~~3. INSTRUCTION STORE .... 12~~
~~4. SEQUENCER INSTRUCTIONS .... 14~~
~~5. CONSTANT STORES .... 14~~
~~6. LOOPING AND BRANCHES .... 15~~
~~6.1 The controlling state .... 15~~
~~6.2 The Control Flow Program .... 15~~
~~6.3 Data dependant predicate instructions .... 18~~
~~6.4 HW Detection of PV,PS .... 18~~
~~6.5 Register file indexing .... 18~~
~~6.6 Predicated Instruction support for Texture clauses .... 19~~
~~6.7 Debugging the Shaders .... 19~~
~~6.7.1 Method 1: Debugging registers .... 19~~
~~6.7.2 Method 2: Exporting the values in the GPRs (12) 19~~
~~7. PIXEL KILL MASK .... 20~~
~~8. HOS SURFACES .... 20~~
~~9. REGISTER FILE ALLOCATION .... 20~~
~~10. FETCH ARBITRATION .... 21~~
~~11. ALU ARBITRATION .... 21~~
~~12. HANDLING STALLS .... 22~~
~~13. CONTENT OF THE RESERVATION STATION FIFOS22~~
~~14. THE OUTPUT FILE .... 22~~
~~15. IJ FORMAT .... 22~~
~~16. THE PARAMETER CACHE .... 23~~
~~17. VERTEX POSITION EXPORTING .... 23~~
~~18. EXPORTING ARBITRATION .... 23~~
~~19. SPECIAL INTERPOLATION MODES .... 24~~
~~19.1 Real time commands .... 24~~
~~19.2 Sprites/ XY screen coordinates/ FB information .... 24~~
~~20. STATE MANAGEMENT .... 25~~
~~21. XY ADDRESS IMPORTS .... 25~~
~~22. REGISTERS .... 25~~
~~22.1 Control .... 25~~
~~22.2 Context .... 25~~
~~23. DEBUG REGISTERS .... 26~~
~~23.1 Context .... 26~~
~~24. INTERFACES .... 26~~
~~24.1 External Interfaces .... 26~~
~~24.1.1 SC to SP : IJ bus .... 27~~
~~24.1.2 SC to SEQ : IJ Control bus .... 27~~
~~24.1.3 SQ to SP: Interpolator bus .... 28~~
~~24.1.4 Sequencer to SPx: Parameter cache write control 28~~
~~24.1.5 SQ to SP: Parameter Cache Read control bus 28~~
~~24.1.6 SQ to SX: Parameter Cache Mux control Bus 29~~
~~24.1.7 SP to SX: Parameter data .... 29~~
~~24.1.8 SX to Interpolators: Parameter Cache Return bus 29~~
~~24.1.9 VGT to SP0/SQ : Vertex Bus .... 29~~
~~24.1.10 SEQ to CP: State report .... 33~~
~~24.1.11 SP to SX : Pixel/Vertex write to SX .. 33~~
~~24.1.12 SQ to SX: Control bus .... 33~~
~~24.1.13 SX to SQ : Output file control .... 34~~
~~24.1.14 Shader Engine to Fetch Unit Bus ..... 34~~
~~24.1.15 Sequencer to Fetch Unit bus .... 34~~
~~24.1.16 Sequencer to SP: GPR control .... 35~~
~~24.1.17 Sequencer to SPx: Instructions .... 35~~
~~24.1.18 SP to Sequencer: Constant address load 36~~
~~24.1.19 Sequencer to SPx: constant broadcast 36~~
~~24.1.20 SP0 to Sequencer: Kill vector load ... 36~~
~~24.1.21 SQ to CP: RBBM bus .... 36~~
~~24.1.22 CP to SQ: RBBM bus .... 36~~
~~25. EXAMPLES OF PROGRAM EXECUTIONS .. 36~~
~~25.1.1 Sequencer Control of a Vector of Vertices 36~~
~~25.1.2 Sequencer Control of a Vector of Pixels 37~~
~~25.1.3 Notes .... 38~~
~~26. OPEN ISSUES .... 39~~
~~1. OVERVIEW .... 4~~
~~1.1 Top Level Block Diagram .... 6~~
~~1.2 Data Flow graph .... 8~~
~~1.3 Control Graph .... 9~~
~~2. INTERPOLATED DATA BUS .... 9~~
~~3. INSTRUCTION STORE .... 12~~
~~4. SEQUENCER INSTRUCTIONS .... 12~~
~~5. CONSTANT STORE .... 12~~
~~6. LOOPING AND BRANCHES .... 12~~
~~6.1 The controlling state .... 12~~
~~6.2 The Control Flow Program .... 13~~
~~6.3 Data dependant predicate instructions .... 15~~
~~6.4 HW Detection of PV,PS .... 15~~
~~6.5 Register file indexing .... 16~~
~~6.6 Predicated Instruction support for Texture clauses .... 16~~
~~6.7 Debugging the Shaders .... 16~~
~~6.7.1 Method 1: Debugging registers .... 16~~
~~6.7.2 Method 2: Exporting the values in the GPRs (12) 17~~
~~6.7.3 Method 3: Selective export of a 32 bit Dword. 17~~
~~7. PIXEL KILL MASK .... 17~~
~~8. HOS SURFACES .... 17~~
~~9. REGISTER FILE ALLOCATION .... 17~~
~~10. FETCH ARBITRATION .... 18~~
~~11. ALU ARBITRATION .... 18~~
~~12. HANDLING STALLS .... 19~~
~~13. CONTENT OF THE RESERVATION STATION FIFOS19~~
~~14. THE OUTPUT FILE .... 19~~
~~15. IJ FORMAT .... 19~~
~~16. THE PARAMETER CACHE .... 20~~
~~17. VERTEX POSITION EXPORTING .... 20~~
~~18. EXPORTING ARBITRATION .... 20~~
~~19. REAL TIME COMMANDS .... 20~~
~~20. REGISTERS .... 21~~
~~20.1 Control .... 21~~
~~20.2 Context .... 21~~
~~21. DEBUG REGISTERS .... 22~~

~~21.1 Control..22~~
~~21.2 Context...22~~
~~**22. INTERFACES..22**~~
~~22.1 External Interfaces ....22~~
~~22.1.1 PA/SC to SP0 : IJ bus ....22~~
~~22.1.2 PA/SC to SEQ : IJ Control bus.........22~~
~~22.1.3 SEQ to SP0 : Interpolator bus..........23~~
~~22.1.4 SEQ to SP0 : Parameter Cache bus 23~~
~~22.1.5 SEQ to SX0 : Parameter Cache Mux control Bus 23~~
~~22.1.6 SX0 to SP0 : Parameter Cache Return bus 23~~
~~22.1.7 VGT to SP0/SEQ : Vertex Bus.........23~~
~~22.1.8 CP to SEQ : Constant store load .....24~~
~~22.1.9 CP to SEQ : Fetch State store load .24~~
~~22.1.10 CP to SEQ : Control State store load 24~~
~~22.1.11 MH to SEQ: Instruction store Load ..24~~
~~22.1.12 SP0 to SX0 : Pixel read from RBs....24~~
~~22.1.13 SEQ to SX0 : Control bus.................24~~
~~22.1.14 SX0 to SEQ : Output file control.......24~~
~~22.1.15 SP0 to SX0 : Position return bus......25~~
~~22.1.16 Shader Engine to Fetch Unit Bus (Fast Bus) 25~~
~~22.1.17 Sequencer to Fetch Unit bus (Slow Bus) 25~~
~~**23. EXAMPLES OF PROGRAM EXECUTIONS..25**~~
~~23.1.1 Sequencer Control of a Vector of Vertices 26~~
~~23.1.2 Sequencer Control of a Vector of Pixels 27~~
~~23.1.3 Notes ....28~~
~~**24. OPEN ISSUES..28**~~

Revision Changes:

Rev 0.1 (Laurent Lefebvre) Date: May 7, 2001	First draft.
Rev 0.2 (Laurent Lefebvre) Date : July 9, 2001	Changed the interfaces to reflect the changes in the SP. Added some details in the arbitration section.
Rev 0.3 (Laurent Lefebvre) Date : August 6, 2001	Reviewed the Sequencer spec after the meeting on August 3, 2001.
Rev 0.4 (Laurent Lefebvre) Date : August 24, 2001	Added the dynamic allocation method for register file and an example (written in part by Vic) of the flow of pixels/vertices in the sequencer.
Rev 0.5 (Laurent Lefebvre) Date : September 7, 2001	Added timing diagrams (Vic)
Rev 0.6 (Laurent Lefebvre) Date : September 24, 2001	Changed the spec to reflect the new R400 architecture. Added interfaces.
Rev 0.7 (Laurent Lefebvre) Date : October 5, 2001	Added constant store management, instruction store management, control flow management and data dependant predication.
Rev 0.8 (Laurent Lefebvre) Date : October 8, 2001	Changed the control flow method to be more flexible. Also updated the external interfaces.
Rev 0.9 (Laurent Lefebvre) Date : October 17, 2001	Incorporated changes made in the 10/18/01 control flow meeting. Added a NOP instruction, removed the conditional_execute_or_jump. Added debug registers.
Rev 1.0 (Laurent Lefebvre) Date : October 19, 2001	Refined interfaces to RB. Added state registers.
Rev 1.1 (Laurent Lefebvre) Date : October 26, 2001	Added SEQ→SP0 interfaces. Changed delta precision. Changed VGT→SP0 interface. Debug Methods added.
Rev 1.2 (Laurent Lefebvre) Date : November 16, 2001	Interfaces greatly refined. Cleaned up the spec.
Rev 1.3 (Laurent Lefebvre) Date : November 26, 2001	Added the different interpolation modes.
Rev 1.4 (Laurent Lefebvre) Date : December 6, 2001	Added the auto incrementing counters. Changed the VGT→SQ interface. Added content on constant management. Updated registers.

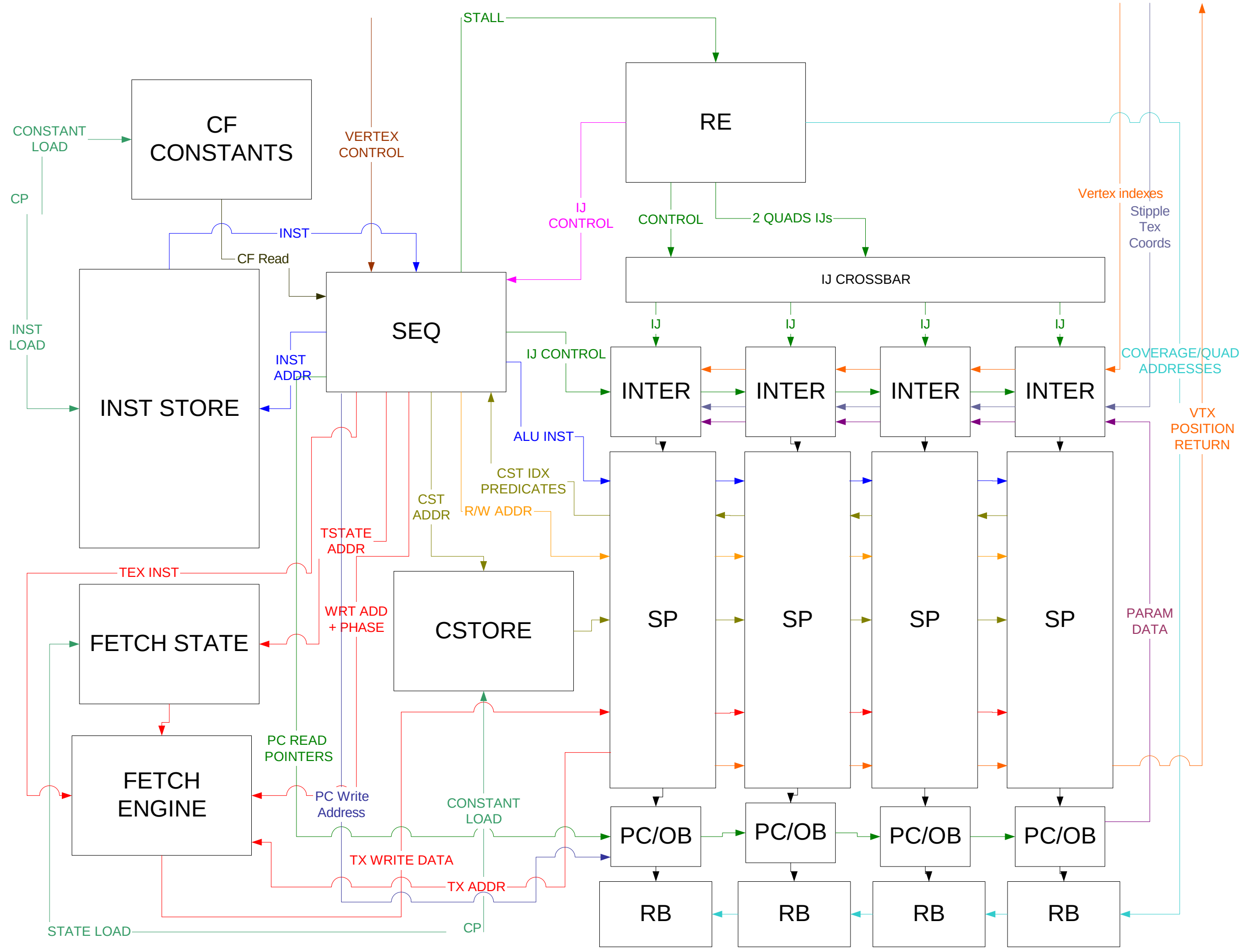
1. Overview

The sequencer is based on the R300 design. It chooses two ALU clauses and a fetch clause to execute, and executes all of the instructions in a clause before looking for a new clause of the same type. Two ALU clauses are executed interleaved to hide the ALU latency. Each vector will have eight fetch and eight ALU clauses, but clauses do not need to contain instructions. A vector of pixels or vertices ping-pongs along the sequencer FIFO, bouncing from fetch reservation station to alu reservation station. A FIFO exists between each reservation stage, holding up vectors until the vector currently occupying a reservation station has left. A vector at a reservation station can be chosen to execute. The sequencer looks at all eight alu reservation stations to choose an alu clause to execute and all eight fetch stations to choose a fetch clause to execute. The arbitrator will give priority to clauses/reservation stations closer to the bottom of the pipeline. It will not execute an alu clause until the fetch fetches initiated by the previous fetch clause have completed. There are two separate sets of reservation stations, one for pixel vectors and one for vertices vectors. This way a pixel can pass a vertex and a vertex can pass a pixel.

To support the shader pipe the sequencer also contains the shader instruction cache, constant store, control flow constants and texture state. The four shader pipes also execute the same instruction thus there is only one sequencer for the whole chip.

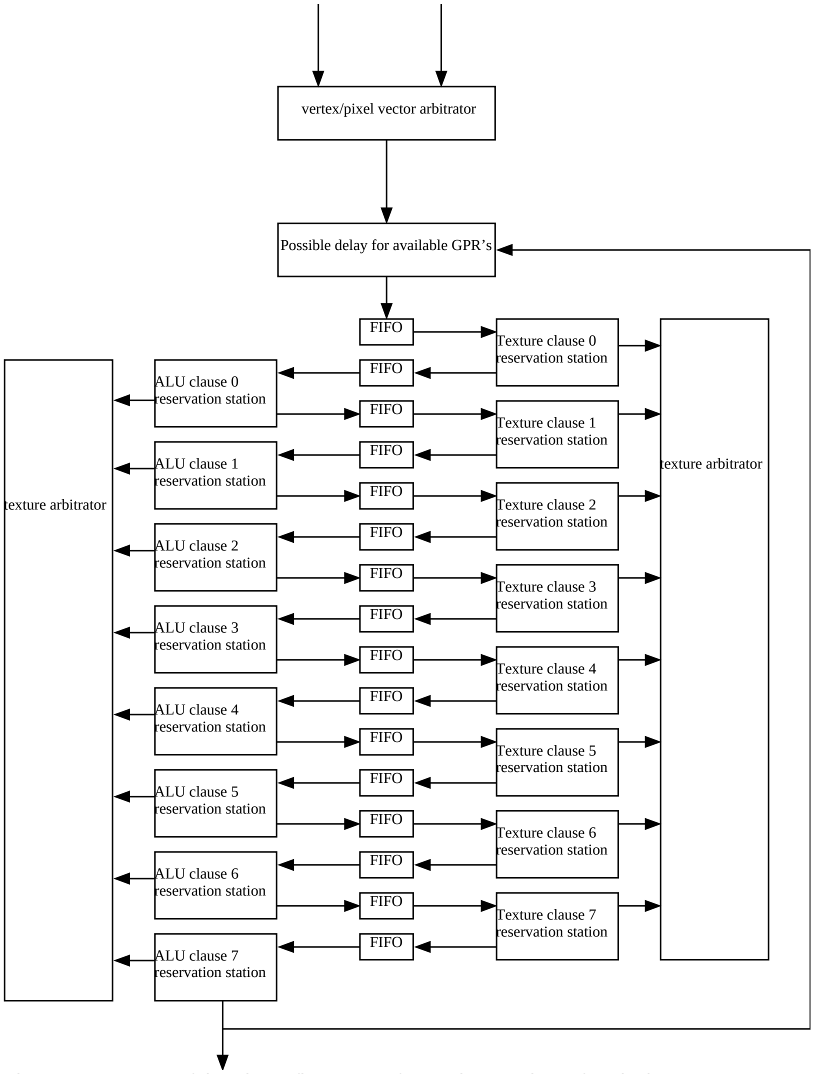
The sequencer first arbitrates between vectors of 64 vertices that arrive directly from primitive assembly and vectors of 16 quads (64 pixels) that are generated in the scan converter.

The vertex or pixel program specifies how many GPRs it needs to execute. The sequencer will not start the next vector until the needed space is available in the GPRs.



STALL
CF CONSTANTS
CONSTANT LOAD
CP
VERTEX CONTROL
RE
IJ CONTROL
CONTROL
2 QUADS IJs
INST
CF Read
SEQ
IJ CROSSBAR
Vertex indexes
Stipple
Tex
Coords
INST LOAD
INST STORE
INST ADDR
IJ
IJ
IJ
IJ
IJ CONTROL
INTER
INTER
INTER
INTER
COVERAGE/QUAD ADDRESSES
VTX POSITION RETURN
ALU INST
CST IDX PREDICATES
CST ADDR
R/W ADDR
TSTATE ADDR
SP
SP
SP
SP
TEX INST
FETCH STATE
WRT ADD + PHASE
CSTORE
PARAM DATA
PC READ POINTERS
FETCH ENGINE
PC Write Address
CONSTANT LOAD
PC/OB
PC/OB
PC/OB
PC/OB
TX WRITE DATA
TX ADDR
RB
RB
RB
RB
STATE LOAD
CP

1.1 Top Level Block Diagram


vertex/pixel vector arbitrator
Possible delay for available GPR's
FIFO
Texture clause 0 reservation station
Texture clause 1 reservation station
Texture clause 2 reservation station
Texture clause 3 reservation station
Texture clause 4 reservation station
Texture clause 5 reservation station
Texture clause 6 reservation station
Texture clause 7 reservation station
texture arbitrator
ALU clause 0 reservation station
ALU clause 1 reservation station
ALU clause 2 reservation station
ALU clause 3 reservation station
ALU clause 4 reservation station
ALU clause 5 reservation station
ALU clause 6 reservation station
ALU clause 7 reservation station
texture arbitrator

There are two sets of the above figure, one for vertices and one for pixels.

Depending on the arbitration state, the sequencer will either choose a vertex or a pixel packet. The control packet consists of 3 bits of state, 7 bits for the base address of the Shader program and some information on the coverage to determine fetch LOD plus other various small state bits.

On receipt of a packet, the input state machine (not pictured but just before the first FIFO) allocated enough space in the registers to store the interpolated values and temporaries. Following this, the barycentric coordinates (and XY screen position if needed) are sent to the interpolator buffers which are going to use these barycentric coordinates to interpolate the parameters and place the interpolated values into the GPRs. Then, the input state machine stacks the packet in the first FIFO.

On receipt of a command, the level 0 fetch machine issues a texture request and corresponding register address for the fetch address (ta). A small command (tcmd) is passed to the fetch system identifying the current level number (0) as well as the register write address for the fetch return data. One fetch request is sent every 4 clocks causing the texturing of sixteen 2x2s worth of data (or 64 vertices). Once all the requests are sent the packet is put in FIFO 1.

Upon receipt of the return data, the fetch unit writes the data to the register file using the write address that was provided by the level 0 fetch machine and sends the clause number (0) to the level 0 fetch state machine to signify that the write is done and thus the data is ready. Then, the level 0 fetch machine increments the counter of FIFO 1 to signify to the ALU 1 that the data is ready to be processed.

On receipt of a command, the level 0 ALU machine first decrements the input FIFO counter and then issues a complete set of level 0 shader instructions. For each instruction, the state machine generates 3 source addresses, one destination address (3 cycles later) and an instruction. Once the last instruction as been issued, the packet is put into FIFO 2.

There will always be two active ALU clauses at any given time (and two arbiters). One arbiter will arbitrate over the odd instructions (4 clocks cycles) and the other one will arbitrate over the even instructions (4 clocks cycles). The only constraints between the two arbiters is that they are not allowed to pick the same clause number as the other one is currently working on if the packet is not of the same type (render state).

If the packet is a vertex packet, upon reaching ALU clause 3, it can export the position if the position is ready. So the arbiter must prevent ALU clause 3 to be selected if the positional buffer is full (or can't be accessed). Along with the positional data, if needed the sprite size and/or edge flags can also be sent.

{ISSUE: How do we handle parameter cache pointers (computed, semi-computed or not computed)?}

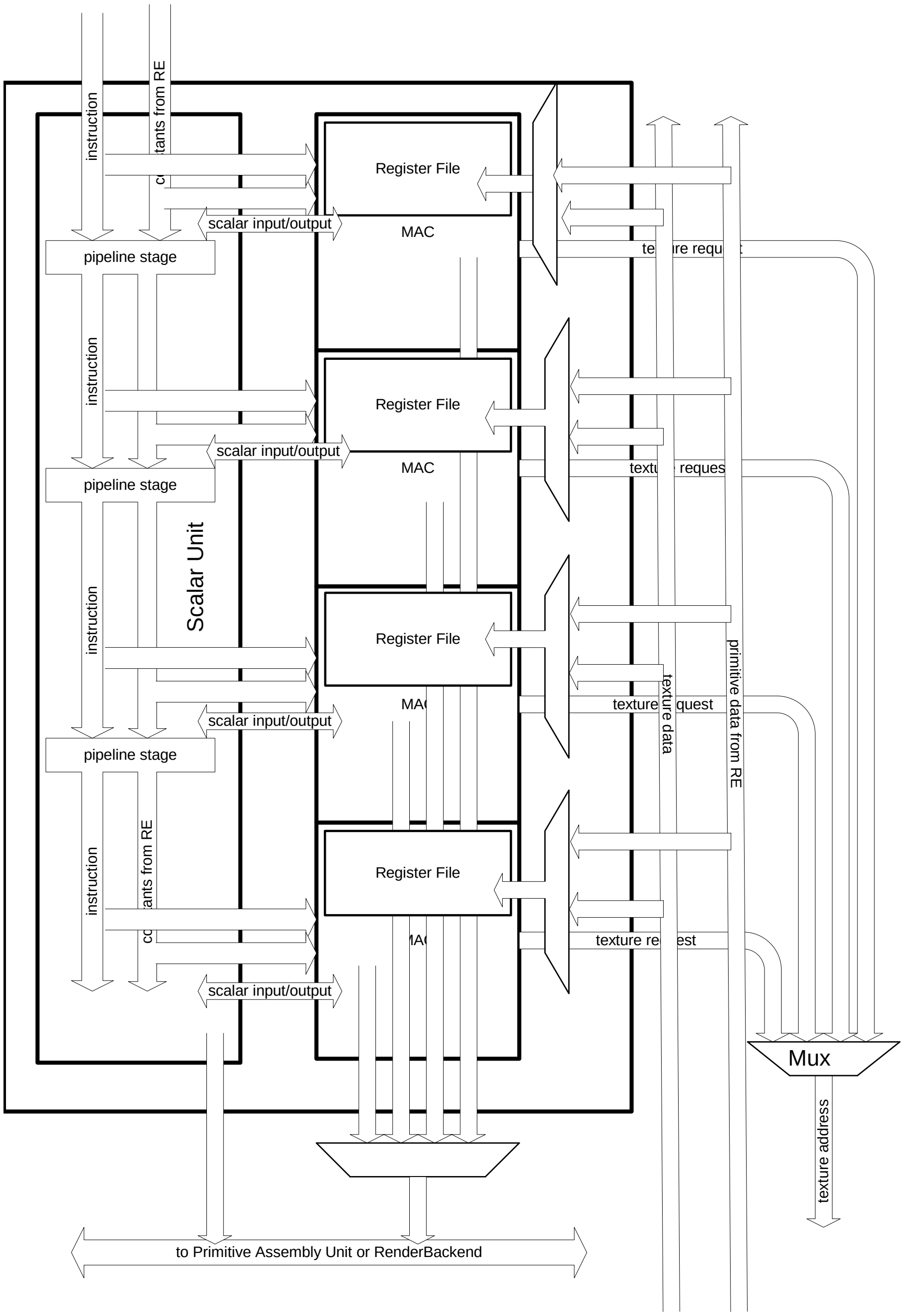
A special case is for multipass vertex ~~shaders which~~shaders, which can export 12 parameters per last 6 clauses to the output buffer. If the output buffer is full or doesn't have enough space the sequencer will prevent such a vertex group to enter an exporting clause.

Multipass pixel shaders can export 12 parameters to memory from the last clause only (7).

All other level process in the same way until the packet finally reaches the last ALU machine (7). ~~Upon completion of a vertex shader, a bit is sent to the SC to let it know that it can begin sending pixels of this group to the sequencer.~~

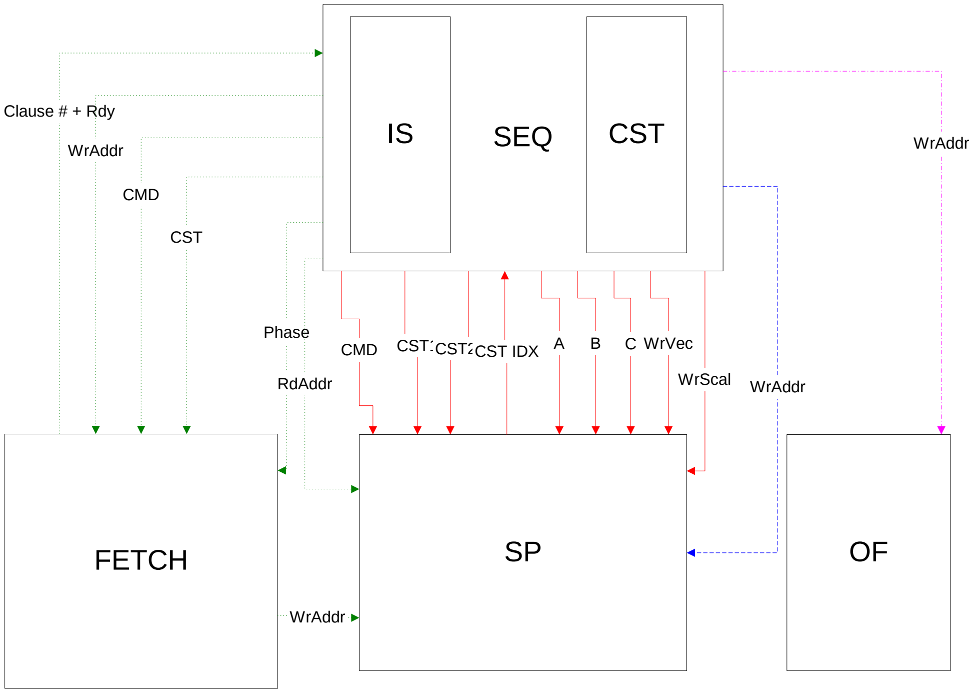
Only two ALU state machine may have access to the register file address bus or the instruction decode bus at one time. Similarly, only one fetch state machine may have access to the register file address bus at one time. Arbitration is performed by three arbiter blocks (two for the ALU state machines and one for the fetch state machines). The arbiters always favor the higher number state machines, preventing a bunch of half finished jobs from clogging up the register files.

1.2 Data Flow graph



The gray area represents blocks that are replicated 4 times per shader pipe (16 times on the overall chip).

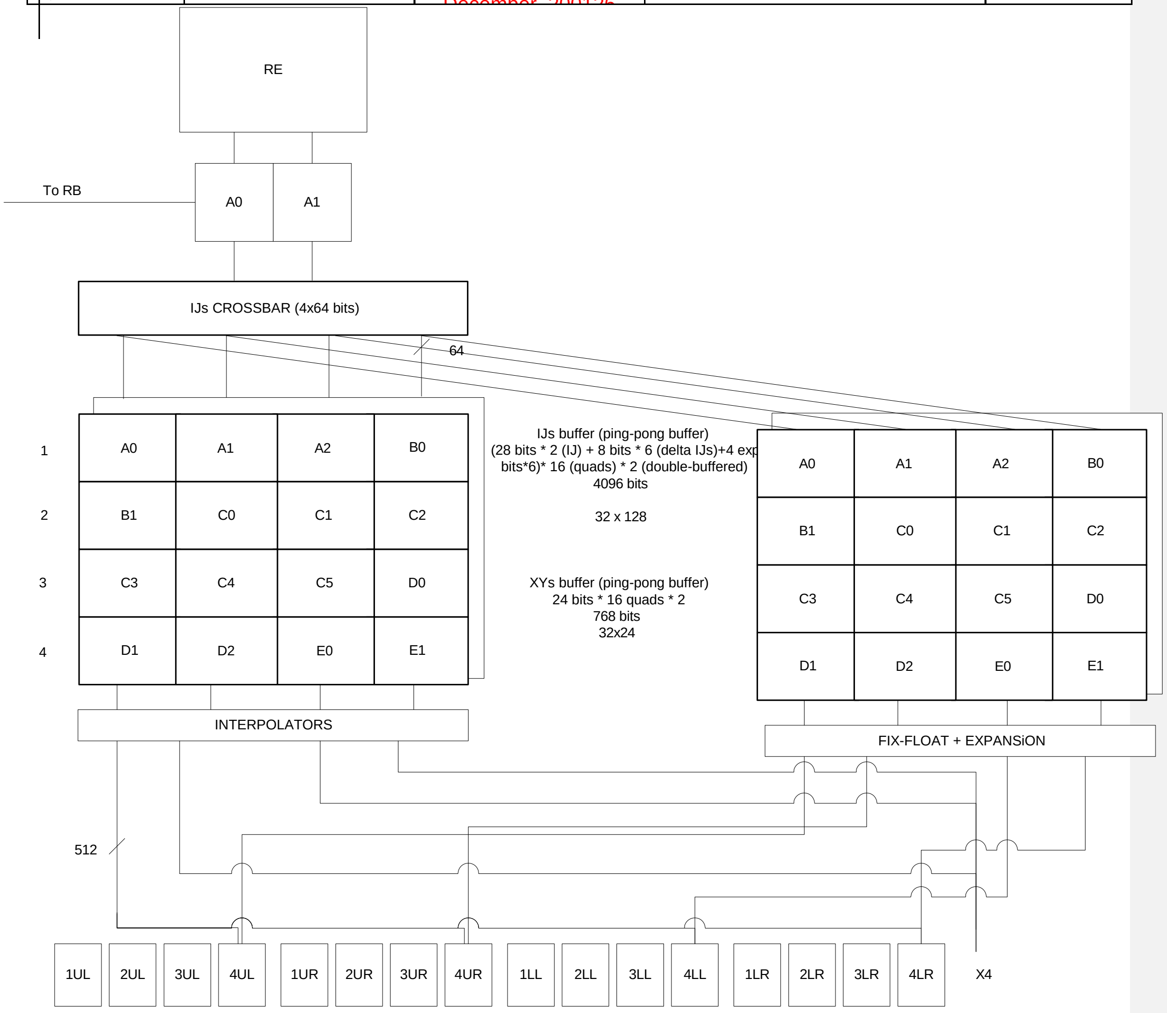
1.3 Control Graph



In green is represented the Fetch control interface, in red the ALU control interface, in blue the Interpolated/Vector control interface and in purple is the output file control interface.

2. Interpolated data bus

The interpolators contain an IJ buffer to pack the information as much as possible before writing it to the register file.



IJs buffer (ping-pong buffer)
(28 bits * 2 (IJ) + 8 bits * 6 (delta IJs)+4 exp
bits*6)* 16 (quads) * 2 (double-buffered)
4096 bits

32 x 128

XYs buffer (ping-pong buffer)
24 bits * 16 quads * 2
768 bits
32x24

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21	T22	T23
SP 0	XY 0-3	XY 16-19	XY 32-35	XY 48-51	A0	B1	C3	D1					A0	B1	C3	D1					V 0-3	V 16-19	V 32-35	V 48-51
SP 1	XY 4-7	XY 20-23	XY 36-39	XY 52-55	A1		C4	D2		C0			A1		C4	D2		C0			V 4-7	V 20-23	V 36-39	V 52-55
SP 2	XY 8-11	XY 24-27	XY 40-43	XY 56-59	A2		C5			C1		E0	A2		C5			C1		E0	V 8-11	V 24-27	V 40-43	V 56-59
SP 3	XY 12-15	XY 28-31	XY 44-47	XY 60-63					B0	C2	D0	E1					B0	C2	D0	E1	V 12-15	V 28-31	V 44-47	V 60-63
	XY				P1								P2								VTX			

Above is an example of a tile we might receive. The IJ information is packed in the IJ buffer 2 quads at a time. The sequencer allows at any given time as many as four quads to interpolate a parameter. They all have to come from the same primitive. Then the sequencer controls the write mask to the register to write the valid data in.

3. Instruction Store

There is going to be only one instruction store for the whole chip. It will contain 4096 instructions of 96 bits each.

It is likely to be a 1 port memory; we use 1 clock to load the ALU instruction, 1 clocks to load the Fetch instruction, 1 clock to load 2 control flow instructions and 1 clock to write instructions.

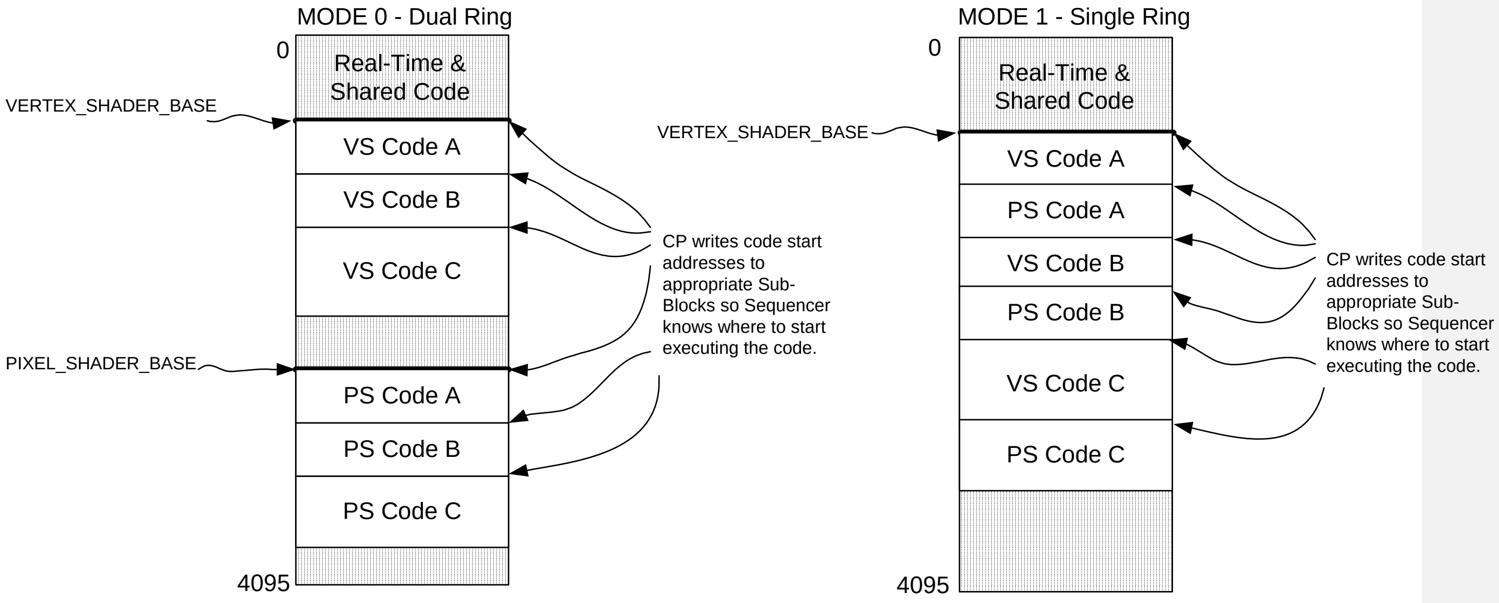
The instruction store is loaded by the CP thru the INST_DATA~~INSTRUCTION_DATA~~, INST_INDEX_PORT ~~INSTRUCTION_INDEX_PORT~~ control registers. The INST_INDEX_PORT~~INSTRUCTION_INDEX_PORT~~ is auto-incremented on both reads and writes to the INST_DATA~~INSTRUCTION_DATA~~ register.

The next picture shows the various modes the CP can load the memory. The Sequencer has to keep track of the loading modes in order to wrap around the correct boundaries. The MSB of the INST_INDEX_PORT ~~INSTRUCTION_INDEX_PORT~~ register contains the packet type for the sequencer to know where it must wrap around. The wrap around points are arbitrary and they are specified in the ~~VERTEX_SHADER~~VS_BASE and PI~~XEL_SHADER~~X_BASE registers.

For the Real time commands the story is quite the same but for some small differences. The CP will use the INST_INDEX_PORT_RT and INST_DATA_RT register pair instead of the regular ones and there are no wrap around points for real time so the driver must be careful not to overwrite regular shader data. The shared code (shared subroutines) uses the same path as real time.

R400 CP's Views of Instruction Memory

Updated: 11/14/2001
John A. Carey



CP writes code start addresses to appropriate Sub-Blocks so Sequencer knows where to start executing the code.

CP writes code start addresses to appropriate Sub-Blocks so Sequencer knows where to start executing the code.

4. Sequencer Instructions

All control flow instructions and move instructions are handled by the sequencer only. The ALUs will perform NOPs during this time (MOV PV,PV, PS,PS).

5. Constant Stores

5.1 Memory organizations

Formatted: Bullets and Numbering

~~The sequencer is aware of where the constants are using a remaping table.~~ A likely size for the ALU constant store is 1024x128 bits. The read BW from the ALU constant store is 128 bits/clock and the write bandwidth is 32 bits/clock (directed by the CP bus size not by memory ports)~~32/4 bits/clock~~.

The maximum logical size of the constant store for a given shader is 256 constants. Or 512 for the pixel/vertex shader pair. The size of the remaping table is 128 lines (each line addresses 4 constants). The write granularity is 4 constants or 512 bits. It takes 16 clocks to write the four constants.

The texture state is also kept in a similar memory. The size of this memory is 192x128. The memory thus holds 128 texture states (192 bits per state). The logical size exposed 32 different states total, which are going to be shared between the pixel and the vertex shader. The size of the remaping table to for the texture state memory is 16 lines (each line addresses 2 texture state lines in the real memory). The write granularity is 2 texture state lines (or 384 bits). The driver sends 512 bits but the CP ignores the top 128 bits. It thus takes 12 clocks to write the two texture states.

The control flow constant memory doesn't sit behind a renaming table. It is register mapped and thus the driver must reload its content each time there is a state change. Its size is 256*32 because it must hold 8 copies of the 32 dwords of control flow constants.

The CP is loading the constant store using the CONST_DATA and CONST_ADDR registers. It does so by writing to the CONST_ADDR register the logical address for the constant block it wants to update and then writes 16 times to the CONST_DATA register. The CONST_ADDR is auto-incremented on both reads and writes to the CONST_DATA register.

5.2 Management of the remaping tables

Formatted: Bullets and Numbering

The sequencer is responsible to manage two remaping tables (one for the constant store and one for the texture state). On a state change (by the driver), the sequencer will broadside copy the contents of its remaping tables to a new one. We have 8 different remaping tables we can use concurrently. More details and a diagram to come….

~~5.1~~5.3 Constant Store Indexing

Formatted: Bullets and Numbering

In order to do constant store indexing, the sequencer must be loaded first with the indexes (that come from the GPRs). There are 144 wires from the exit of the SP to the sequencer (9 bits pointers x 16 vertexes/clock). Since the data must pass thru the Shader pipe for the float to fixed conversion, there is a latency of 4 clocks (1 instruction) between the time the sequencer is loaded and the time one can index into the constant store. The assembly will look like this

```
MOVA  R1.X,R2.X       // Loads the sequencer with the content of R2.X, also copies the content of R2.X into R1.X
NOP                   // latency of the float to fixed conversion
ADD   R3,R4,C0[R2.X]// Uses the state from the sequencer to add R4 to C0[R2.X] into R3
```

Note that we don't really care about what is in the brackets because we use the state from the MOVA instruction. R2.X is just written again for the sake of simplicity and coherency.

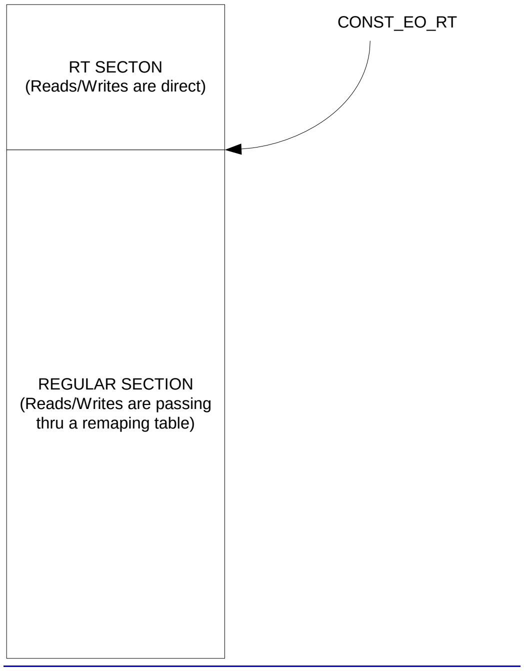
The storage needed in the sequencer in order to support this feature is 2*64*9 bits = 1152 bits.

Formatted: Bullets and Numbering

5.4

5.5 Real Time Commands

The real time commands constants are written by the CP using the CONST_DATA_RT and CONST_ADDR_RT registers. It works is the same way than when dealing with regular constant loads BUT in this case the CP is not sending a logical address but rather a physical address and the reads are not passing thru the remaping table but are directly read from the memory. The boundary between the two zones is defined by the CONST_EO_RT control register.


CONST_EO_RT
RT SECTON
(Reads/Writes are direct)
REGULAR SECTION
(Reads/Writes are passing thru a remaping table)

~~The texture state is also kept in a similar memory. The size of this memory is 192x128. Which lets us load a texture state in 2 clocks. The memory thus holds 96 texture states (2*128 bits per state)~~

~~The control flow constant memory doesn't sit behind a renaming table. It is register mapped and thus the driver must reload its content each time there is a state change. Its size is 192*32 because it must hold 8 copies of the 24 dwords of control flow constants.~~

6. Looping and Branches

Loops and branches are planned to be supported and will have to be dealt with at the sequencer level. We plan on supporting constant loops and branches using a control program.

6.1 The controlling state.

The R400 controling state consists of:
~~As per Dx9 the following state is available for control flow:~~

~~Boolean[15:0]~~
~~loop_count[7:0][7:0]~~
~~In addition:~~

~~loop_start [7:0] [7:0]~~
~~loop_step [7:0] [7:0]~~
~~Exist to give more control to the controlling program.~~

~~We will extend that in the R400 to:~~
Boolean[~~255~~256:0]
Loop_count[7:0][~~31:0~~31:0]
Loop_Start[7:0]~~ ~~[~~[31:0~~31:0]
Loop_Step[7:0]~~ ~~[~~31:~~31:0]
~~Loop_End[7:0] [31:0]~~

That is 256 Booleans and 32 loops.

We have a stack of 4 elements for nested call~~ings~~ of subroutines and 4 loop counters to allow for nested loops.

6.2 The Control Flow Program

~~The R300 uses a match method for control flow: The shader is executed, and at every instruction its address is compared with addresses (or address?) in a control table. The "event" in the control table can redirect operations in the program.~~
The basic model is as follows:

The render state defined the clause boundaries:
Vertex_shader_fetch[7:0][7:0] // eight 8 bit pointers to the location where each clauses control program is located
Vertex_shader_alu[7:0][7:0] // eight 8 bit pointers to the location where each clauses control program is located
Pixel_shader_fetch[7:0][7:0] // eight 8 bit pointers to the location where each clauses control program is located
Pixel_shader_alu[7:0][7:0] // eight 8 bit pointers to the location where each clauses control program is located

A pointer value of FF means that the clause doesn't contain any instructions.

The control program for a given clause is executed to completion before moving to another clause, (with the exception of the pick two nature of the alu execution). The control program is the only program aware of the clause boundaries.

~~The Method chosen for the R400 is a "control program".~~ The control program has ~~ten~~ eleven basic instructions:

Execute
Conditional_execute
Conditional_Execute_Predicates
Conditional_jump
Call
Return
Loop_start
Loop_end
End_of_clause
Conditional_End_of_clause
NOP

Execute, causes the specified number of instructions in instruction store to be executed.
Conditional_execute checks a condition first, and if true, causes the specified number of instructions in instruction store to be executed.
Loop_start resets the corresponding loop counter to the start value on the first pass after it checks for the end condition and if met jumps over to a specified address.
Loop_end increments (decrements?) the loop counter and jumps back the specified number of instructions.
Call jumps to an address and pushes the IP counter on the stack. On the return instruction, the IP is popped from the stack.
Conditional_execute_or_Jump executes a block of instructions or jumps to an address is the condition is not met.
Conditional_execute_Predicates executes a block of instructions if all bits in the predicate vectors meet the condition.
End_of_clause marks the end of a clause.

Conditional_End_of_clause marks the end of a clause if the condition is met.
Conditional_jumps jumps to an address if the condition is met.
NOP is a regular NOP

NOTE THAT ALL JUMPS MUST JUMP TO EVEN CFP ADDRESSES. Thus the compiler must insert NOPs where needed to align the jumps on even CFP addresses.

Also if the jump is logically bigger than pshader_cntl_size (or vshader_cntl_size) we break the program (clause) and set the debug registers. If an execute or conditional_execute is lower than cntl_size or bigger than size we also break the program (clause) and set the debug registers.

We have to fit instructions into 48 bits in order to be able to put two control flow instruction per line in the instruction store.

Note that whenever a field is marked as RESERVED, it is assumed that all the bits of the field are cleared (0).

Formatted

Execute				
47	46… 42	41 … 24	23 … 12	11 … 0
Addressing	00001	RESERVED	Instruction -count	Exec Address

Execute up to 4k instructions at the specified address in the instruction memory.

NOP		
47	46 … 42	41 … 0
Addressing	00010	RESERVED

This is a regular NOP.

Conditionnal_Execute							
47	46 … 42	41~~41 … 34~~	40 … 33	~~33~~32	~~32~~ 31 … 24	23 … 12	11 … 0
Addressing	00011	RESERVED ~~Boolean address~~	Boolean address	Condition	RESERVED	Instruction -count	Exec Address

If the specified boolean (8 bits can address 256 booleans) meets the specified condition then execute the specified instructions (up to 4k instructions)

Conditionnal_Execute_Predicates							
47	46 … 42	41 … 35	~~41~~ 34 … ~~38~~33	~~37~~32	~~36~~ 31 … 24	23 … 12	11 … 0
Addressing	00100	RESERVED	Predicate vector	Condition	RESERVED	Instruction- count	Exec Address

Check the AND/OR of all current predicate bits. If AND/OR matches the condition execute the specified number of instructions. We need to AND/OR this with the kill mask in order not to consider the pixels that aren't valid.

Loop_Start				
47	46 … 42	41 … 17	~~16 … 5~~16 … 12	~~4 … 0~~11 … 0
Addressing	00101	RESERVED	~~Jump address~~loop ID	~~Loop ID~~Jump address

Loop Start. Compares the loop count with the end value. If loop condition not met jump to the address. Forward jump only. Also computes the index value.

Loop_End				
47	46 … 42	41 … 17	16 … ~~5~~12	~~4 … 0~~11 … 0

Addressing	00111	RESERVED	~~Start address~~loop ID	~~Loop ID~~start address

Loop end. Increments the counter by one and jumps BACK only to the start of the loop.

The way this is described does not prevent nested loops, and the inclusion of the loop id make this easy to do.

Call			
47	46 … 42	41…12	11 … 0
Addressing	01000	RESERVED	Jump address~~Address~~

Jumps to the specified address and pushes the IP counter on the stack.

Return		
47	46 … 42	41 … 0
Addressing	01001	RESERVED

Pops the topmost address from the stack and jumps to that address. If nothing is on the stack, the program will just continue to the next instruction.

Conditionnal_Jump							
47	46 … 42	41~~41 … 34~~	40 … 33	~~33~~32	~~32 … 13~~31	~~12~~30 … 12	11 … 0
Addressing	01010	RESERVED ~~Boolean address~~	Boolean address	Condition	FW only~~RESERVED~~	RESERVED~~FW only~~	Jump address~~Address~~

If condition met, jumps to the address. FORWARD jump only allowed if bit ~~12~~ 31 set. Bit ~~12~~ 31 is only an optimization for the compiler and should NOT be exposed to the API.

Conditional_End_of_Clause					
47	46 … 42	41~~41 … 34~~	40 … 33	~~33~~32	~~32~~ 31 … 0
Addressing	01011	RESERVED ~~Boolean address~~	Boolean address	Condition	RESERVED

This is an optimization in the case of very short shaders (where the control flow instruction can't be hidden anymore and thus are not free. In this case, if the condition is met, the clause is ended, else we continue the execution of the clause.

End_of_Clause		
47	46 … 42	41 … 0
Addressing	01011	RESERVED

Marks the end of a clause.

To prevent infinite loops, we will keep 9 bits loop counters instead of 8 (we are only able to loop 256 times). If the counter goes higher than 255 then the loop_end or the loop_start instruction is going to break the loop and set the debug registers.

~~The basic model is as follows:~~

~~The render state defined the clause boundaries:~~
~~Vertex_shader_fetch[7:0][7:0] // eight 8 bit pointers to the location where each clauses control program is located~~
~~Vertex_shader_alu[7:0][7:0] // eight 8 bit pointers to the location where each clauses control program is located~~
~~Pixel_shader_fetch[7:0][7:0] // eight 8 bit pointers to the location where each clauses control program is located~~
~~Pixel_shader_alu[7:0][7:0] // eight 8 bit pointers to the location where each clauses control program is located~~

~~**A pointer value of FF means that the clause doesn't contain any instructions.**~~

~~The control program for a given clause is executed to completion before moving to another clause, (with the exception of the pick two nature of the alu execution). The control program is the only program aware of the clause boundaries.~~

6.3 Data dependant predicate instructions

Data dependant conditionals will be supported in the R400. The only way we plan to support those is by supporting three vector/scalar predicate operations of the form:

PRED_SETE_# - similar to SETE except that the result is 'exported' to the sequencer.
PRED_SETNE_# - similar to SETNE except that the result is 'exported' to the sequencer.
PRED_SETGT_# - similar to SETGT except that the result is 'exported' to the sequencer
PRED_SETGTE_# - similar to SETGTE except that the result is 'exported' to the sequencer

For the scalar operations only we will also support the two following instructions:
PRED_SETE0_# – SETE0
PRED_SETE1_# – SETE1

The export is a single bit - 1 or 0 that is sent using the same data path as the MOVA instruction. The sequencer will maintain 4 sets of 64 bit predicate vectors (in fact 8 sets because we interleave two programs but only 4 will be exposed) and use it to control the write masking. This predicate is not maintained across clause boundaries. The # sign is used to specify which predicate set you want to use 0 thru 3.

Then we have two conditional execute bits. The first bit is a conditional execute "on" bit and the second bit tells us if we execute on 1 or 0. For example, the instruction:

P0_ADD_# R0,R1,R2

Is only going to write the result of the ADD into those GPRs whose predicate bit is 0. Alternatively, P1_ADD_# would only write the results to the GPRs whose predicate bit is set. The use of the P0 or P1 without precharging the sequencer with a PRED instruction is undefined.

{Issue: do we have to have a NOP between PRED and the first instruction that uses a predicate?}

6.4 HW Detection of PV,PS

Because of the control program, the compiler cannot detect statically dependant instructions. In the case of non-masked writes and subsequent reads the sequencer will insert uses of PV,PS as needed. This will be done by comparing the read address and the write address of consecutive instructions. For masked writes, the sequencer will insert NOPs wherever there is a dependant read/write.

The sequencer will also have to insert NOPs between PRED_SET and MOVA instructions and their uses.

6.5 Register file indexing

Because we can have loops in fetch clause, we need to be able to index into the register file in order to retrieve the data created in a fetch clause loop and use it into an ALU clause. The instruction will include the base address for register indexing and the instruction will contain these controls:

Bit7	Bit 6	
0	0	'absolute register'
0	1	'relative register'
1	0	'previous vector'
1	1	'previous scalar'

In the case of an absolute register we just take the address as is. In the case of a relative register read we take the base address and we add to it the loop_index and this becomes our new address that we give to the shader pipe.

The sequencer is going to keep a loop index computed as such:

Index = Loop_counter*Loop_iterator + Loop_init.

The index is going to return 0 if it is out of the range.

6.6 Predicated Instruction support for Texture clauses

For texture clauses, we support the following optimization: we keep 1 bit (thus 4 bits for the four predicate vectors) per predicate vector in the reservation stations. A value of 1 means that one ore more elements in the vector have a value of one (thus we have to do the texture fetches for the whole vector. A value of 0 means that no elements in the vector have his predicate bit set and we can thus skip over the texture fetch. **We have to make sure the invalid pixels aren't considered with this optimization.**

6.7 Debugging the Shaders

In order to be able to debug the pixel/vertex shaders efficiently, we provide 2 methods.

6.7.1 *Method 1: Debugging registers*

Current plans are to expose 2 debugging, or error notification, registers:
1. address register where the first error occurred
2. count of the number of errors

The sequencer will detect the following groups of errors:
- count overflow
- jump error
  relative jump address > size of the control flow program
  relative jump address > length of the shader program
- constant overflow
- register overflow
- call stack
  call with stack full
  return with stack empty

With two of the errors, a jump error or a register overflow will cause the program to break. In this case, a break means that a clause will halt execution, but allowing further clauses to be executed.

With the other errors, program can continue to run, potentially to worst-case limits.

If indexing outside of the constant range, causing an overflow error, the hardware is specified to return the value with an index of 0. This could be exploited to generate error tokens, by reserving and initializing the 0th register (or constant) for errors.

{ISSUE : Interrupt to the driver or not?}

6.7.2 *Method 2: Exporting the values in the GPRs (12)*

The sequencer will have a count register and an address register for this mode and 3 bits per clause specifying the execution mode for each clause. The modes can be :

1) Normal
2) Debug Kill
3) Debug Addr + Count

Under the normal mode execution follows the normal course. Under the kill mode, all control flow instructions are executed but all normal shader instructions of the clause are replaced by NOPs. Only debug_export instructions of clause 7 will be executed under the debug kill setting. Under the other mode, normal execution is done until we reach an address specified by the address register and instruction count (useful for loops) specified by the count register. After we have hit the instruction n times (n=count) we switch the clause to the kill mode.

Under the debug mode (debug kill OR debug Addr + count), it is assumed that clause 7 is always exporting 12 debug vectors and that all other exports to the SX block (position, color, z, ect) will been turned off (changed into NOPs) by the sequencer (even if they occur before the address stated by the ADDR debug register).

7. Pixel Kill Mask

A vector of 64 bits is kept by the sequencer per group of pixels/vertices. Its purpose is to optimize the texture fetch requests and allow the shader pipe to kill pixels using the following instructions:

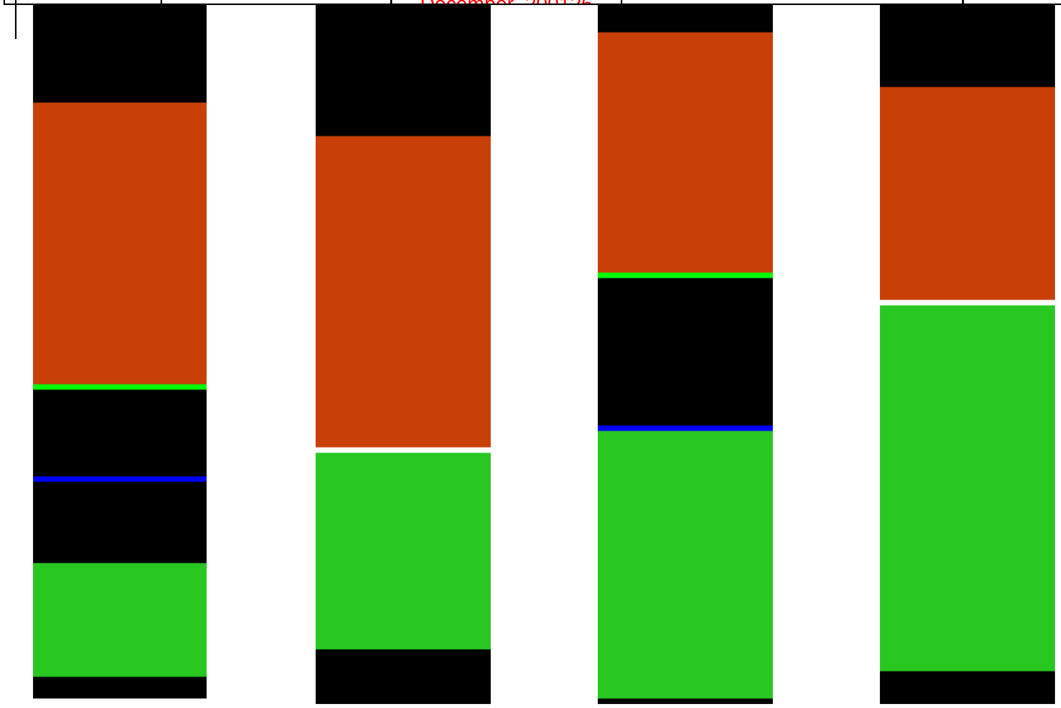
MASK_SETE
MASK_SETNE
MASK_SETGT
MASK_SETGTE

8. ~~HOS surfaces~~Multipass vertex shaders (HOS)

~~HOS~~Multipass vertex shaders~~ surfaces~~ are able to export from the 6 last clauses but to memory ONLY.~~ If they want to export to the parameter cache they have to do it in the last clause (7). They can also export position in clause 3.~~

9. Register file allocation

The register file allocation for vertices and pixels can either be static or dynamic. In both cases, the register file in managed using two round robins (one for pixels and one for vertices). In the dynamic case the boundary between pixels and vertices is allowed to move, in the static case it is fixed to VERTEX_REG_SIZE for vertices and 256-VERTEX_REG_SIZE for pixels.



Above is an example of how the algorithm works. Vertices come in from top to bottom; pixels come in from bottom to top. Vertices are in orange and pixels in green. The blue line is the tail of the vertices and the green line is the tail of the pixels. Thus anything between the two lines is shared. When pixels meets vertices the line turns white and the boundary is static until both vertices and pixels share the same “unallocated bubble”. Then the boundary is allowed to move again.

10. Fetch Arbitration

The fetch arbitration logic chooses one of the 8 potentially pending fetch clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute. Once chosen, the clause state machine will send one 2x2 fetch per clock (or 4 fetches in one clock every 4 clocks) until all the fetch instructions of the clause are sent. This means that there cannot be any dependencies between two fetches of the same clause.

The arbitrator will not wait for the fetches to return prior to selecting another clause for execution. The fetch pipe will be able to handle up to X(?) in flight fetches and thus there can be a fair number of active clauses waiting for their fetch return data.

11. ALU Arbitration

ALU arbitration proceeds in almost the same way than fetch arbitration. The ALU arbitration logic chooses one of the 8 potentially pending ALU clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute. There are two ALU arbitrers, one for the even clocks and one for the odd clocks. For exemple, here is the sequencing of two interleaved ALU clauses (E and O stands for Even and Odd sets of 4 clocks):

Einst0 Oinst0 Einst1 Oinst1 Einst2 Oinst2 Einst0 Oinst3 Einst1 Oinst4 Einst2 Oinst0…
Proceeding this way hides the latency of 8 clocks of the ALUs.

12. Handling Stalls

When the output file is full, the sequencer prevents the ALU arbitration logic to select the last clause (this way nothing can exit the shader pipe until there is place in the output file. If the packet is a vertex packet and the position buffer is full (POS_FULL) then the sequencer also prevents a thread to enter the exporting clause (3?). The sequencer will set the OUT_FILE_FULL signal n clocks before the output file is actually full and thus the ALU arbiter will be able read this signal and act accordingly by not preventing exporting clauses to proceed.

13. Content of the reservation station FIFOs

~~21~~ 3 bits of Render State 7 bits for the base address of the GPRs, some bits for LOD correction and coverage mask information in order to fetch fetch for only valid pixels, quad address and 1 bit to specify if the vector is of pixels or vertices. Since pixels and vertices are kept in order in the shader pipe, we only need two fifos (one for vertices and one for pixels) deep enough to cover the shader pipe latency. This size will be determined later when we will know the size of the small fifos between the reservation stations.

14. The Output File

The output file is where pixels are put before they go to the RBs. The write BW to this store is 256 bits/clock. Just before this output file are staging registers with write BW 512 bits/clock and read BW 256 bits/clock. The staging registers are 4x128 (and there are 16 of those on the whole chip).

15. IJ Format

The IJ information sent by the PA is of this format on a per quad basis:

We have a vector of IJ's (one IJ per pixel at the centroid of the fragment or at the center of the pixel depending on the mode bit). The interpolation is done at a different precision across the 2x2. The upper left pixel's parameters are always interpolated at full 20x24 mantissa precision. Then the result of the interpolation along with the difference in IJ in reduced precision is used to interpolate the parameter for the other three pixels of the 2x2. Here is how we do it:

Assuming P0 is the interpolated parameter at Pixel 0 having the barycentric coordinates I(0), J(0) and so on for P1,P2 and P3. Also assuming that A is the parameter value at V0 (interpolated with I), B is the parameter value at V1 (interpolated with J) and C is the parameter value at V2 (interpolated with (1-I-J).

$$\Delta 01I = I(1) - I(0)$$
$$\Delta 01J = J(1) - J(0)$$
$$\Delta 02I = I(2) - I(0)$$
$$\Delta 02J = J(2) - J(0)$$
$$\Delta 03I = I(3) - I(0)$$
$$\Delta 03J = J(3) - J(0)$$

P0	P1
P2	P3

$$P0 = C + I(0) * (A - C) + J(0) * (B - C)$$
$$P1 = P0 + \Delta 01I * (A - C) + \Delta 01J * (B - C)$$
$$P2 = P0 + \Delta 02I * (A - C) + \Delta 02J * (B - C)$$
$$P3 = P0 + \Delta 03I * (A - C) + \Delta 03J * (B - C)$$

P0 is computed at 20x24 mantissa precision and P1 to P3 are computed at 8X24 mantissa precision. So far no visual degradation of the image was seen using this scheme.

Multiplies (Full Precision): 2
Multiplies (Reduced precision): 6

Subtracts 19x24 (Parameters): 2
Adds: 8

FORMAT OF P0's IJ : Mantissa 20 Exp 4 for I + Sign
Mantissa 20 Exp 4 for J + Sign

FORMAT of Deltas (x3):Mantissa 8 Exp 4 for I + Sign
Mantissa 8 Exp 4 for J + Sign

Total number of bits : ~~19~~20*2 + ~~8*6 + 4*8 + 4*2~~8*6 + 4*8 + 4*2 = 128

The Deltas have a leading 1, the Full precision IJs don't. This means that in the case of the deltas we MUST be able to shift 8 right (exponent value of 0 means number = 0, exponent value of 1 means shift right 8~~8~~). This means that the maximum range for the IJs (Full precision) is +/- 63~~4~~ and the range for the Deltas is +/- 12~~8~~7.

15.1 Interpolation of constant attributes

Formatted: Bullets and Numbering

Because of the floating point imprecision, we need to take special provisions if all the interpolated terms are the same or if two of the barycentric coordinates are the same.

We start with the premise that if A = B and B = C and C = A, then P0,1,2,3 = A. Since one or more of the IJ terms may be zero, so we extend this to:

```
if (A=B and B=C and C=A)
   P0,1,2,3 = A;
else if ((I = 0) or (J = 0)) and
          ((J = 0) or (1-I-J = 0)) and
          ((1-J-I = 0) or (I = 0))) {
                if(I != 0) {
                    P0 = A;
                } else if(J != 0) {
                    P0 = B;
                } else {
                    P0 = C;
               }
             //rest of the quad interpolated normally
}
else
{
             normal interpolation
}
```

16. The parameter cache

The parameter cache is where the vertex shaders export their data. It consists of 16 128x128 memories (1R/1W). The reuse engine will make it so that all vertexes of a given primitive will hit different memories.

17. Vertex position exporting

On clause 3 the vertex shader can export to the PA both the vertex position and the point sprite. It can also do so at clause 7 if not done at clause 3. The storage needed to perform the position export is at least 64x128 memories for the position and 64x32 memories for the sprite size. It is going to be taken in the pixel output fifo from the SX blocks. The clause where the position export occurs is specified by the EXPORT_LATE register. If turned on, it means that the export is going to occur at ALU clause 7 if unset position export occurs at clause 3.

18. Exporting Arbitration

Here are the rules for co-issuing exporting ALU clauses.

1) Position exports and position exports cannot be co-issued.
2) Position exports and memory exports cannot be co-issued.
3) Position exports and Z/Color exports cannot be co-issued.
4) Memory exports and Z/Color exports cannot be co-issued.
5) Memory exports and memory exports cannot be co-issued.
6) Z/color exports and Z/color exports cannot be co-issued.
7) Parameter exports and Z/Color exports CAN be co-issued.
8) Parameter exports and parameter exports CAN be co-issued.
9) Parameter exports and memory exports CAN be co-issued.

Formatted: Bullets and Numbering

19. Export Types

The export type (or the location where the data should be put) is specified using the destination address field in the ALU instruction. Here is a list of all possible export modes:

Formatted: Bullets and Numbering

19.1 Vertex Shading

0:15	- 16 parameter cache
16:31	- Empty (Reserved?)
32:43	- 12 vertex exports to the frame buffer and index
44:47	- Empty
48:59	- 12 debug export (interpret as normal vertex export)
60	- export addressing mode
61	- Empty
62	- sprite size export that goes with position export (point_h,point_w,edgeflag,misc)
63	- position

Formatted: Bullets and Numbering

19.2 Pixel Shading

0	- Color for buffer 0 (primary)
1	- Color for buffer 1
2	- Color for buffer 2
3	- Color for buffer 3
4:7	- Empty
8	- Buffer 0 Color/Fog (primary)
9	- Buffer 1 Color/Fog
10	- Buffer 2 Color/Fog
11	- Buffer 3 Color/Fog
12:15	- Empty
16:31	- Empty (Reserved?)
32:43	- 12 exports for multipass pixel shaders.
44:47	- Empty
48:59	- 12 debug exports (interpret as normal pixel export)
60	- export addressing mode
61:62	- Empty
63	- Z for primary buffer (Z exported to 'alpha' component)

Formatted: Bullets and Numbering

~~19.~~20. Special Interpolation modes

~~19.1~~20.1 Real time commands

We are unable to use the parameter memory since there is no way for a command stream to write into it. Instead we need to add three 16x128 memories (one for each of three vertices x 16 interpolants). These will be mapped onto the register bus and written by type 0 packets, and output to the the parameter busses (the sequencer and/or PA need to be able to address the reatime parameter memory as well as the regular parameter store. For higher performance we should be able able to view them as two banks of 16 and do double buffering allowing one to be loaded, while the other is rasterized with. Most overlay shaders will need 2 or 4 scalar coordinates, one option might be to restrict the memory to 16x64 or 32x64 allowing only two interpolated scalars per cycle, the only problem I see with this is, if we view support for 16 vector-4 interpolants important (true only if we map Microsoft's high priority stream to the realtime stream), then the PA/sequencer need to support a realtime-specific mode where we need to address 32 vectors of parameters instead of 16. This mode is triggered by the primitive type: REAL TIME.

~~19.2~~20.2 Sprites/ XY screen coordinates/ FB information

Formatted: Bullets and Numbering

When working with sprites, one may want to overwrite the parameter 0 with SC generated data. Also, XY screen coordinates may be needed in the shader program. This functionality is controlled by the gen_I0 register (in SQ) in conjunction with the SND_XY register (in SC). Also it is possible to send the faceness information (for OGL front/back special operations) to the shader using the same control registers. Here is a list of all the modes and how they interact together:

Gen_st is a bit taken from the interface between the SC and the SQ. This is the MSB of the primitive type. If the bit is set, it means we are dealing with Point AA, Line AA or sprite and in this case the vertex values are going to ~~be read from the GEN_S and GEN_T state registers instead of being read from the parameter cache~~generated between 0 and 1.

Param_Gen_I0 disable, snd_xy disable, no gen_st – I0 = No modification
Param_Gen_I0 disable, snd_xy disable, gen_st – I0 = No modification
Param_Gen_I0 disable, snd_xy enable, no gen_st – I0 = No modification
Param_Gen_I0 disable, snd_xy enable, gen_st – I0 = No modification
Param_Gen_I0 enable, snd_xy disable, no gen_st – I0 = garbage, garbage, garbage, faceness
Param_Gen_I0 enable, snd_xy disable, gen_st – I0 = garbage, garbage, s, t
Param_Gen_I0 enable, snd_xy enable, no gen_st – I0 = screen x, screen y, garbage, faceness
Param_Gen_I0 enable, snd_xy enable, gen_st – I0 = screen x, screen y, s, t

20.3 Auto generated counters

Formatted: Bullets and Numbering

In the cases we are dealing with multipass shaders, the sequencer is going to generate a vector count to be able to both use this count to write the 1st pass data to memory and then use the count to retrieve the data on the 2nd pass. The count is always generated in the same way but it is passed to the shader in a slightly different way depending on the shader type (pixel or vertex). This is toggled on and off using the GEN_INDEX register. While there is only one count broadcast to the registers, the LSB are hardwired to specific values making the index different for all elements in the vector.

Formatted

Formatted

20.3.1 *Vertex shaders*

Formatted: Bullets and Numbering

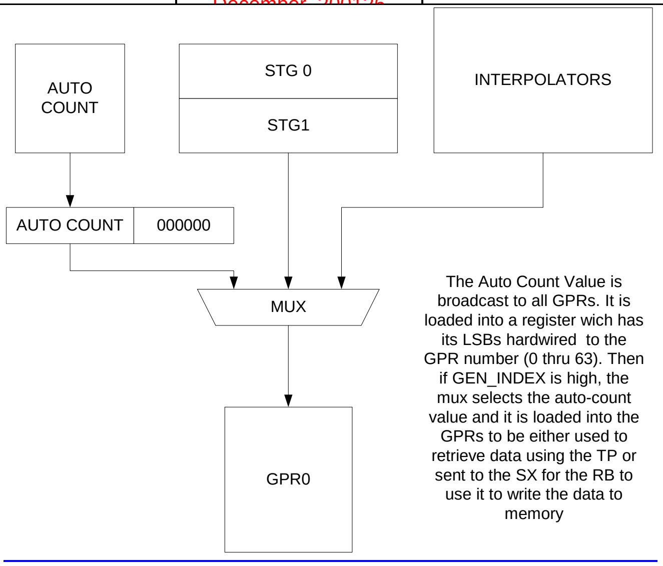
In the case of vertex shaders, if GEN_INDEX is set, the data will be put into the x field of the third register (it means that the compiler must allocate 3 GPRs in all multipass vertex shader modes).

20.3.2 *Pixel shaders*

Formatted: Bullets and Numbering

In the case of pixel shaders, if GEN_INDEX is set, the data will be put in the x field of the 2nd register (I1.x).

Formatted



The Auto Count Value is broadcast to all GPRs. It is loaded into a register wich has its LSBs hardwired to the GPR number (0 thru 63). Then if GEN_INDEX is high, the mux selects the auto-count value and it is loaded into the GPRs to be either used to retrieve data using the TP or sent to the SX for the RB to use it to write the data to memory

Formatted: Bullets and Numbering

~~20.~~21. State management

Every clock, the sequencer will report to the CP the oldest states still in the pipe. These are the states of the programs as they enter the last ALU clause.

Formatted: Bullets and Numbering

~~21.~~22. XY Address imports

The SC will be able to send the XY addresses to the GPRs. It does so by interleaving the writes of the IJs (to the IJ buffer) with XY writes (to the XY buffer). Then when writing the data to the GPRs, the sequencer is going to interpolate the IJ data or pass the XY data thru a Fix→float converter and expander and write the converted values to the GPRs. The Xys are currently SCREEN SPACE COORDINATES. The values in the XY buffers will wrap. See section 20.2~~19.2~~ for details on how to control the interpolation in this mode.

Formatted: Bullets and Numbering

22.1 Vertex indexes imports

In order to import vertex indexes, we have 64x2x96 staging registers. These are loaded one at a time by the VGT block. They are loaded in floating point format and can be transferred in 4 or 8 clocks to the GPRs.

Formatted: Bullets and Numbering

~~22.~~23. Registers

~~22.1~~23.1 Control

~~DYNAMIC_~~REG_DYNAMIC Dynamic allocation (pixel/vertex) of the register file on or off.
~~VERTEX_REG_SIZE What portion of the register file is reserved for vertices (static allocation only)~~
REG_SIZE_PIX~~PIXEL_MIN_SIZE~~ S~~Minimal s~~ize of the register file's pixel portion (minimal size when dynamic allocation turned on) ~~(dynamic only)~~
REG_SIZE_VTX ~~VERTEX_MIN_SIZE Minimal s~~Size of the register file's vertex portion (minimal size when dynamic allocation turned on) ~~(dynamic only)~~
ARBITRATION_~~policy~~POLICY policy of the arbitration between vertexes and pixels
INST_STORE_ALLOC interleaved, separate
~~VERTEX_BASE~~INST_BASE_VTX start point for the vertex instruction store (RT always ends at vertex_base and

Begins at 0)

~~PIXEL_BASE~~INST_BASE_PIX start point for the pixel shader instruction store

~~NO_INTERLEAVE~~ONE_THREAD debug state register. Only allows one program at a time into the GPRs

~~NO_INTERLEAVE_ALU~~ONE_ALU debug state register. Only allows one ALU program at a time to be executed (instead of 2)

INSTRUCTION_INDEX_PORT This is where the CP puts the base address of the instruction writes and type (auto-incremented on reads/writes)

INSTRUCTION_DATA This is where the CP puts the actual data going to the instruction memory

CONSTANT_DATA This is where the CP puts constant data (32 bits)

CONSTANT_ADDR This is where the CP puts the logical constant address (9 bits)

INSTRUCTION_INDEX_PORT_RT This is where the CP puts the base address of the instruction writes and type for Real Time (auto-incremented on reads/writes)

INSTRUCTION_DATA_RT This is where the CP puts the actual data going to the instruction memory for Real Time

CONSTANT_DATA_RT This is where the CP puts constant data for Real Time (32 bits)

CONSTANT_ADDR_RT This is where the CP puts the logical constant address for Real Time (9 bits)

CONSTANT_EO_RT This is the size of the space reserved for real time in the constant store (from 0 to CONSTANT_EO_RT). The remaping table operates on the rest of the memory

EXPORT_LATE Controls whether or not we are exporting position from clause 3. If set, position exports occur at clause 7.

Formatted

~~22.2~~23.2 Context

Formatted: Bullets and Numbering

~~Vshader_fetch[7:0][7:0]~~VS_FETCH_{0…7} eight 8 bit pointers to the location where each clauses control program is located

~~Vshader_alu[7:0][7:0]~~VS_ALU_{0…7} eight 8 bit pointers to the location where each clauses control program is located

PS_FETCH_{0…7} ~~Pshader_fetch[7:0][7:0]~~ eight 8 bit pointers to the location where each clauses control program is located

PS_ALU_{0…7} ~~Pshader_alu[7:0][7:0]~~ eight 8 bit pointers to the location where each clauses control program is located

~~PSHADER~~PS_BASE base pointer for the pixel shader in the instruction store

~~VSHADER~~VS_BASE base pointer for the vertex shader in the instruction store

~~Vshader_cntl_size~~VS_CF_SIZE size of the vertex shader (# of instructions in control program/2)

~~Pshader_cntl_size~~PS_CF_SIZE size of the pixel shader (# of instructions in control program/2)

~~Pshader_size~~PS_SIZE size of the pixel shader (cntl+instructions)

~~Vshader_size~~VS_SIZE size of the vertex shader (cntl+instructions)

~~REG_ALLOC_PIX~~PS_NUM_REG number of registers to allocate for pixel shader programs

~~REG_ALLOC_VERT~~VS_NUM_REG number of registers to allocate for vertex shader programs

~~FLAT_GOUR[0…15]~~PARAM_SHADE One 16 bit register specifying which parameters are to be gouraud shaded (0 = flat, 1 = gouraud)

~~CYL_WRAP[0…63]~~ PARAM_WRAP 64 bits: for which parameters (and channels (xyzw)) do we do the cyl wrapping (0=linear, 1=cylindrical).

~~P_export_mode~~PS_EXPORT_MODE 0xxxx : Normal mode
1xxxx : Multipass mode
If normal, bbbz where bbb is how many colors (0-4) and z is export z or not
If multipass 1-12 exports for color.

~~vshader_export_mask~~VS_EXPORT_MASK which of the last 6 ALU clauses is exporting (multipass only)

~~vshader_export_mode~~VS_EXPORT_MODE 0: position (1 vector), 1: position (2 vectors), 3:multipass

~~vshader_export_count[6]~~VS_EXPORT_COUNT_{0…6} Six 4 bit counters representing the ~~#~~# of interpolated parameters exported in clause 7 (located in VS_EXPORT_COUNT_6) OR
# of exported vectors to memory per clause in multipass mode (per clause)

~~Control_Flow~~ ~~24 Dwords that contain the control flow constants.~~

PARAM_~~GEN_T~~ ~~Max Value interpolated in the T coordinate field (sprites)~~

~~GEN_S~~ ~~Max Value interpolated in the S coordinate field (sprites)~~

GEN_I0 Do we overwrite or not the parameter 0 with XY data and generated T and S values
GEN_INDEX Auto generates an address from 0 to XX. Puts the results into R1 for pixel shaders and R3 for vertex shaders
CONST_BASE_VTX~~VTX_CST_BASE~~ (9 bits) Logical Base address for the constants of the Vertex shader
~~PIX_CST_BASE~~CONST_BASE_PIX (9 bits) Logical Base address for the constants of the Pixel shader
~~PIX_CST_SIZE~~CONST_SIZE_PIX (8 bits) Size of the logical constant store for pixel shaders
~~VTX_CST_SIZE~~CONST_SIZE_VTX (8 bits) Size of the logical constant store for vertex shaders
INST_PRED_OPTIMIZE Turns on the predicate bit optimization (if of, conditional_execute_predicates is always executed).
CF_BOOLEANS 256 boolean bits
CF_LOOP_COUNT 32x8 bit counters (number of times we traverse the loop)
CF_LOOP_START 32x8 bit counters (init value used in index computation)
CF_LOOP_STEP 32x8 bit counters (step value used in index computation)

Formatted: Bullets and Numbering

~~23.~~24. DEBUG registers

~~23.1~~24.1 Context

DB_PROB_ADDR instruction address where the first problem occurred
DB_PROB_COUNT number of problems encountered during the execution of the program
~~Count~~DB_INST_COUNT instruction counter for debug method 2
~~Addr~~DB_BREAK_ADDR break address for method number 2
DB_CLAUSE_MODE_ALU_{0…7}~~Clause_mode[3]~~ clause mode for debug method 2 (0: normal, 1: addr, 2: kill)
DB_CLAUSE_MODE_FETCH_{0…7} clause mode for debug method 2 (0: normal, 1: addr, 2: kill)

~~NO_PRED_OPTIMIZE turns off the predicate bit optimization (conditional_execute_predicates is always executed.~~

Formatted: Bullets and Numbering

~~24.~~25. Interfaces

~~24.1~~25.1 External Interfaces

Whenever an x is used, it means that the bus is broadcast to all units of the same name. For example, if a bus is named SQ→SPx it means that SQ is going to broadcast the same information to all SP instances.

Formatted: Bullets and Numbering

~~24.1.1~~25.1.1 *SC to SP : IJ bus*

This is a bus that sends the IJ information to the IJ fifos on the top of each shader pipe. At the same time the control information goes to the sequencer. There are 4 of these buses over the whole chip (SP0 thru 3)

Name	Direction	Bits	Description
SC_SP0_data	SC→SP0	64	IJ information sent over 2 clocks (or XY info sent over 1 clock in the lower 24 LSBs of the interface)
SC_SP0_q_wr_mask	SC→SP0	1	Write Mask
SC_SP0_dest	SC→SP0	1	Controls the write destination (XY buffer, IJ buffer)
SC_SP1_data	SC→SP1	64	IJ information sent over 2 clocks (or XY info sent over 1 clock in the lower 24 LSBs of the interface)
SC_SP1_q_wr_mask	SC→SP1	1	Write Mask
SC_SP1_dest	SC→SP1	1	Controls the write destination (XY buffer, IJ buffer)
SC_SP2_data	SC→SP2	64	IJ information sent over 2 clocks (or XY info sent over 1 clock in the lower 24 LSBs of the interface)
SC_SP2_q_wr_mask	SC→SP2	1	Write Mask
SC_SP2_dest	SC→SP2	1	Controls the write destination (XY buffer, IJ buffer)
SC_SP3_data	SC→SP3	64	IJ information sent over 2 clocks (or XY info sent over 1 clock in the lower 24 LSBs of the interface)
SC_SP3_q_wr_mask	SC→SP3	1	Write Mask
SC_SP3_dest	SC→SP3	1	Controls the write destination (XY buffer, IJ buffer)

~~24.1.2~~25.1.2 SC to S~~E~~Q : IJ Control bus

Formatted: Bullets and Numbering

This is the control information sent to the sequencer in order to control the IJ fifos and all other information needed to execute a shader program on the sent pixels. This information is sent over 2 clocks, if SENDXY is asserted the next control packet is going to be ignored and XY information is going to be sent on the IJ bus (for the quads that where just sent).

Name	Direction	Bits	Description
SC_SQ_q_wr_mask	SC→SQ	4	Quad Write mask left to right
SC_SQ_lod_correct	SC→SQ	24	LOD correction per quad (6 bits per quad)
SC_SQ_flat_vertex	SC→SQ	2	Provoking vertex for flat shading
SC_SQ_param_ptr0	SC→SQ	11	P Store pointer for vertex 0
SC_SQ_param_ptr1	SC→SQ	11	P Store pointer for vertex 1
SC_SQ_param_ptr2	SC→SQ	11	P Store pointer for vertex 2
SC_SQ_end_of_vect	SC→SQ	1	End of the vector
SC_SQ_store_dealloc	SC→SQ	1	Deallocation token for the P Store
SC_SQ_state	SC→SQ	3	State/constant pointer (6*3+3)
SC_SQ_valid_pixel	SC→SQ	16	Valid bits for all pixels
SC_SQ_null_prim	SC→SQ	1	Null Primitive (for PC deallocation purposes)
SC_SQ_end_of_prim	SC→SQ	1	End Of the primitive
SC_SQ_fbface	SC→SQ	1	Front face = 1, back face = 0
SC_SQ_send_xy	SC→SQ	1	Sending XY information [XY information is going to be sent on the next clock]
SC_SQ_prim_type	SC→SQ	3	Real time command need to load tex cords from alternate buffer. Line AA, Point AA and Sprite reads their parameters from GEN_T and GEN_S registers. 000 : Normal 011 : Real Time 100 : Line AA 101 : Point AA 110 : Sprite
SC_SQ_new_vector	SC→SQ	1	This primitive comes from a new vector of vertices. Make sure that the corresponding vertex shader has finished before starting the group of pixels.
SC_SQ_RTRn	SQ→SC	1	Stalls the PA in n clocks
SC_SQ_RTS	SC→SQ	1	SC ready to send data

~~24.1.3~~25.1.3 SQ to SP: Interpolator bus

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SPx_interp_prim_type	SQ→SPx	3	Type of the primitive 000 : Normal 011 : Real Time 100 : Line AA 101 : Point AA 110 : Sprite
SQ_SPx_interp_flat_vtx	SQ→SPx	2	Provoking vertex for flat shading
SQ_SPx_interp_flat_gouraud	SQ→SPx	1	Flat or gouraud shading
SQ_SPx_interp_cyl_wrap	SQ→SPx	4	Wich parameter needs to be cylindrical wrapped
SQ_SPx_interp_ijline	SQ→SPx	2	Line in the IJ/XY buffer to use to interpolate
SQ_SPx_interp_buff_swap	SQ→SPx	1	Swap the IJ/XY buffers at the end of the interpolation
SQ_SPx_interp_gen_I0	SQ→SPx	1	Generate I0 or not. This tells the interpolators not to use the parameter cache but rather overwrite the data with interpolated 1 and 0. Overwrite if gen_I0 is high.

25.1.4 SQ to SP: GPR Input Mux select

Formatted: Bullets and Numbering

This interface is synchronized with the Interpolator bus. This controls the input mux to the GPRs. The three types of data are: generated index, Interpolated data, vertex index data (coming from the staging registers).

Name	Direction	Bits	Description
SQ_SPx_data_type	SQ→SPx	2	00: Interpolated data 01: Staging register data 1x: Count
SQ_SPx_index_count	SQ→SPx	12?	Index count, common for all shader pipes
SQ_SPx_stage_addr	SQ→SPx	1	Staging register address 0: First staging register 1: second staging register

Formatted

Formatted

Formatted

Formatted

25.1.5 SQ to SPx: Parameter cache write control

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SPx_pc_wr_addr	SQ→SPx	7	Write address
SQ_SPx_pc_we_addr	SQ→SPx	1	Write Enable
SQ_SPx_pc_phase_mux	SQ→SPx	1	The output selector mux (gpr vs parameter cache)

~~24.1.4~~25.1.6 SQ to SP: Parameter Cache Read control bus

Formatted: Bullets and Numbering

The four following interfaces (SQ→SP, SQ→SX,SP→SX and SX→Interpolators) are all SYNCHRONIZED together.

Name	Direction	Bits	Description
SQ_SPx_ptr0	SQ→SPx	9	Pointer of PC
SQ_SPx_ptr1	SQ→SPx	9	Pointer of PC
SQ_SPx_ptr2	SQ→SPx	9	Pointer of PC
SQ_SP0_read_ena	SQ→SP0	4	Read enables for the 4 memories in the SP0
SQ_SP1_read_ena	SQ→SP1	4	Read enables for the 4 memories in the SP1
SQ_SP2_read_ena	SQ→SP2	4	Read enables for the 4 memories in the SP2
SQ_SP3_read_ena	SQ→SP3	4	Read enables for the 4 memories in the SP3

~~24.1.5~~25.1.7 SQ to SX: Parameter Cache Mux control Bus

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SXx_mux0	SQ→SXx	4	Mux control for PC (4 MSbs of Pointer)
SQ_SXx_mux1	SQ→SXx	4	Mux control for PC (4 MSbs of Pointer)
SQ_SXx_mux2	SQ→SXx	4	Mux control for PC (4 MSbs of Pointer)

~~24.1.6~~25.1.8 SP to SX: Parameter data

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SP0_SX0_data0	SP0→SX0	128	Parameter data 0
SP0_SX0_data1	SP0→SX0	128	Parameter data 1
SP0_SX0_data2	SP0→SX0	128	Parameter data 2
SP0_SX0_data3	SP0→SX0	128	Parameter data 3
SP1_SX1_data0	SP1→SX1	128	Parameter data 0
SP1_SX1_data1	SP1→SX1	128	Parameter data 1
SP1_SX1_data2	SP1→SX1	128	Parameter data 2
SP1_SX1_data3	SP1→SX1	128	Parameter data 3
SP2_SX0_data0	SP2→SX0	128	Parameter data 0
SP2_SX0_data1	SP2→SX0	128	Parameter data 1
SP2_SX0_data2	SP2→SX0	128	Parameter data 2
SP2_SX0_data3	SP2→SX0	128	Parameter data 3
SP3_SX1_data0	SP3→SX1	128	Parameter data 0
SP3_SX1_data1	SP3→SX1	128	Parameter data 1
SP3_SX1_data2	SP3→SX1	128	Parameter data 2

SP3_SX1_data3	SP3→SX1	128	Parameter data 3

~~24.1.7~~25.1.9 *SX to Interpolators: Parameter Cache Return bus*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SXx_SPx_vtx_data_0	SXx→SPx	128	Vertex data to interpolate
SXx_SPx_vtx_data_1	SXx→SPx	128	Vertex data to interpolate
SXx_SPx_vtx_data_2	SXx→SPx	128	Vertex data to interpolate

25.1.10 *SQ to SP0: Staging Register Data*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SP0_vgt_vsisr_data	SQ→SP0	96	Pointers of indexes or HOS surface information
SQ_SP0_vgt_vsisr_double	SQ→SP0	1	0: Normal 96 bits per vert 1: double 192 bits per vert
SQ_SP0_data_valid	SQ→SP0	1	Data is valid

25.1.11 *PA to SQ : Vertex interface*

Formatted: Bullets and Numbering

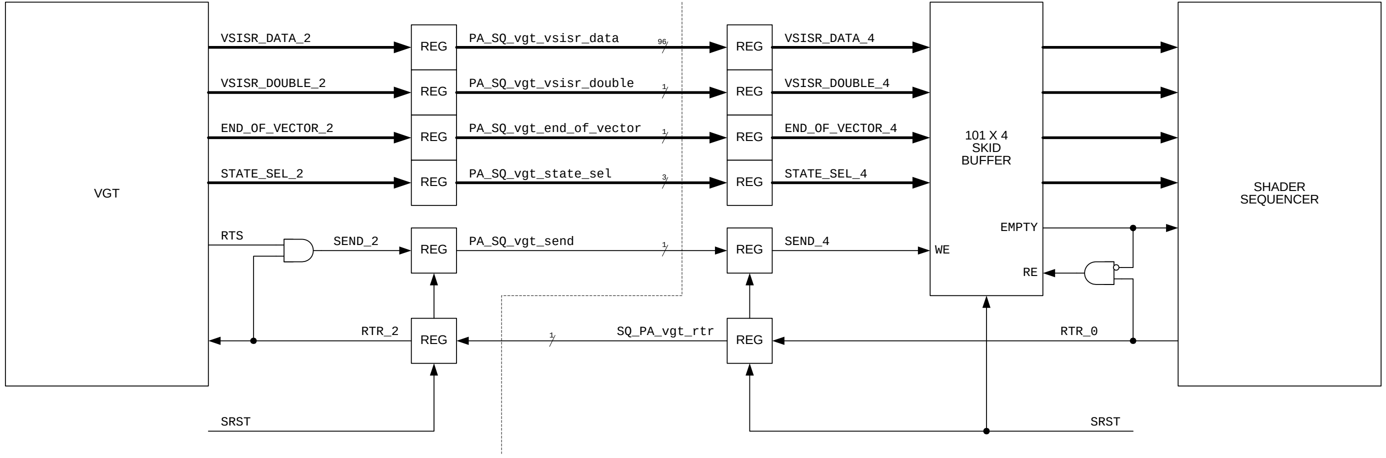
25.1.11.1 Interface Signal Table

The area difference between the two methods is not sufficient to warrant complicating the interface or the state requirements of the VSISRs. **Therefore, the POR for this interface is that the VGT will transmit the data to the VSISRs (via the Shader Sequencer) in full, 32-bit floating-point format.** The VGT can transmit up to six 32-bit floating-point values to each VSISR where four or more values require two transmission clocks. The data bus is 96 bits wide.

Name	Bits	Description
PA_SQ_vgt_vsisr_data	96	Pointers of indexes or HOS surface information
PA_SQ_vgt_vsisr_double	1	0: Normal 96 bits per vert 1: double 192 bits per vert
PA_SQ_vgt_end_of_vector	1	Indicates the last VSISR data set for the current process vector (for double vector data, "end_of_vector" is set on the second vector)
PA_SQ_vgt_state	3	Render State (6*3+3 for constants). This signal is guaranteed to be correct when "PA_SQ_vgt_end_of_vector" is high.
PA_SQ_vgt_send	1	Data on the VGT_SQ is valid receive (see write-up for standard R400 SEND/RTR interface handshaking)
SQ_PA_vgt_rtr	1	Ready to receive (see write-up for standard R400 SEND/RTR interface handshaking)

25.1.11.2 Interface Diagrams

Formatted: Bullets and Numbering



VGT
VSISR_DATA_2
VSISR_DOUBLE_2
END_OF_VECTOR_2
STATE_SEL_2
RTS
SEND_2
RTR_2
SRST
REG
PA_SQ_vgt_vsisr_data
96
PA_SQ_vgt_vsisr_double
1
PA_SQ_vgt_end_of_vector
1
PA_SQ_vgt_state_sel
3
PA_SQ_vgt_send
1
SQ_PA_vgt_rtr
1
VSISR_DATA_4
VSISR_DOUBLE_4
END_OF_VECTOR_4
STATE_SEL_4
SEND_4
101 X 4
SKID
BUFFER
EMPTY
WE
RE
RTR_0
SRST
SHADER
SEQUENCER

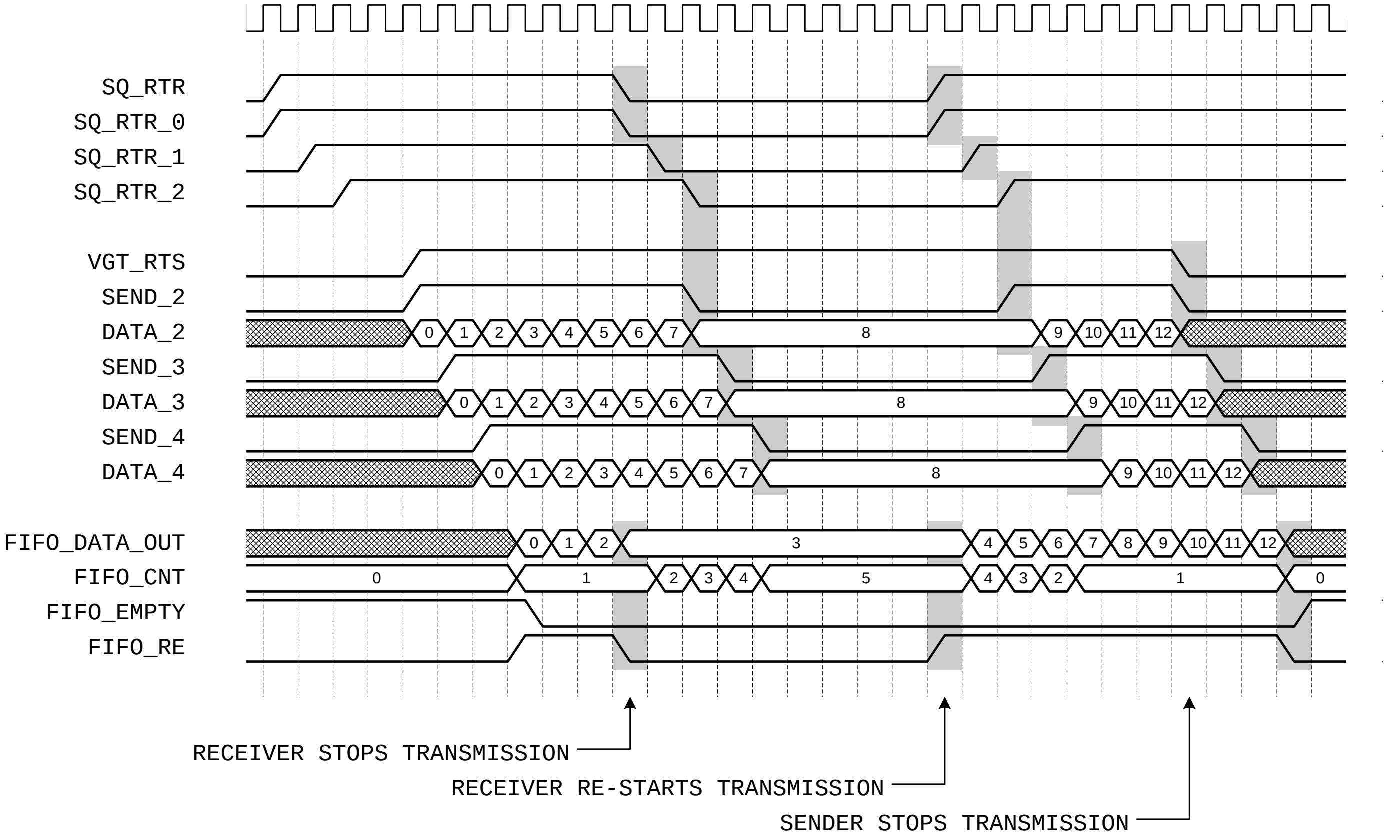


Figure 1. Detailed Logical Diagram for PA_SQ_vgt Interface.

~~24.1.8 VGT to SP0/SQ : Vertex Bus~~

~~24.1.9~~

Formatted: Bullets and Numbering

~~24.1.10~~25.1.12 S~~E~~Q to CP: State report

Name	Direction	Bits	Description
SQ_CP_vrtx_ state	SEQ→CP	3	Oldest vertex state still in the pipe
SQ_CP_pix_state	SEQ→CP	3	Oldest pixel state still in the pipe

~~24.1.10~~25.1.13 SP to SX : Pixel/Vertex write to SX

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SP0_SX0_Export_data	SP0→SX0	256	4 pairs of 32 bits channel values
SP0_SX0_Shader_Dest	SP0→SX0	4	Specifies one of the of up to 12 export destinations
SP1_SX1_Export_data	SP1→SX1	256	4 pairs of 32 bits channel values
SP1_SX1_Shader_Dest	SP1→SX1	4	Specifies one of the of up to 12 export destinations
SP2_SX0_Export_data	SP2→SX0	256	4 pairs of 32 bits channel values
SP2_SX0_Shader_Dest	SP2→SX0	4	Specifies one of the of up to 12 export destinations
SP3_SX1_Export_data	SP3→SX1	256	4 pairs of 32 bits channel values
SP3_SX1_Shader_Dest	SP3→SX1	4	Specifies one of the of up to 12 export destinations
SPx_SXx_Shader_Count	SP0→SX0	3	Each set of four pixels or vectors is exported over eight clocks. This field specifies where the SP is in that sequence.
SPx_SXx_Shader_Last	SP0→SX0	1	Asserted on the first shader count of the last export of the clause
SP0_SX0_Shader_PixelValid	SP0→SX0	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SP0_SX0_Shader_WordValid	SP0→SX0	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors
SP1_SX1_Shader_PixelValid	SP1→SX1	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SP1_SX1_Shader_WordValid	SP1→SX1	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors
SP2_SX0_Shader_PixelValid	SP2→SX0	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SP2_SX0_Shader_WordValid	SP2→SX0	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors
SP3_SX1_Shader_PixelValid	SP3→SX1	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SP3_SX1_Shader_WordValid	SP3→SX1	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors

~~24.1.11~~25.1.14 SQ to SX: Control bus

Formatted: Bullets and Numbering

Name	Direction	Bits	Description

SQ_SXx_exp_Pixel	SQ→SXx	1	1: Pixel 0: Vertex
SQ_SXx_exp_start	SQ→SXx	1	Raised to indicate that the SQ is starting an export
SQ_SXx_exp_Clause	SQ→SXx	3	Clause number, which is needed for vertex clauses
SQ_SXx_exp_State	SQ→SXx	3	State ID, which is needed for vertex clauses

These fields are sent synchronously with SP export data, described in SP0→SX0 interface
{ISSUE: Where are the PC pointers}

~~24.1.12~~25.1.15 *SX to SQ : Output file control*

Formatted: Bullets and Numbering

Name	**Direction**	**Bits**	**Description**
SXx_SQ_Export_count_rdy	SXx→SQ	1	Raised by SX0 to indicate that the following two fields reflect the result of the most recent export
SXx_SQ_Export_Position	SXx→SQ	1	Specifies whether there is room for another position.
SXx_SQ_Export_Buffer	SXx→SQ	7	Specifies the space available in the output buffers. 0: buffers are full 1: 2K-bits available (32-bits for each of the 64 pixels in a clause) ... 64: 128K-bits available (16 128-bit entries for each of 64 pixels) 65-127: RESERVED

~~24.1.13~~25.1.16 *Shader Engine to Fetch Unit Bus*

Formatted: Bullets and Numbering

Four quad's worth of addresses is transferred to Fetch Unit every clock. These are sourced from a different pixel within each of the sub-engines repeating every 4 clocks. The register file index to read must precede the data by 2 clocks. The Read address associated with Quad 0 must be sent 1 clock after the Instruction Start signal is sent, so that data is read 3 clocks after the Instruction Start.

Four Quad's worth of Fetch Data may be written to the Register file every clock. These are directed to a different pixel of the sub-engines repeating every 4 clocks. The register file index to write must accompany the data. Data and Index associated with the Quad 0 must be sent 3 clocks after the Instruction Start signal is sent.

Name	**Direction**	**Bits**	**Description**
SP0_TP0_fetch_addr	SP0->TP0	512	4 Fetch Addresses read from the Register file
TP0_SP0_data	TP0→SP0	512	4 texture results
SP1_TP1_fetch_addr	SP1->TP1	512	4 Fetch Addresses read from the Register file
TP1_SP1_data	TP1→SP1	512	4 texture results
SP2_TP2_fetch_addr	SP2->TP2	512	4 Fetch Addresses read from the Register file
TP2_SP2_data	TP2→SP2	512	4 texture results
SP3_TP3_fetch_addr	SP3->TP3	512	4 Fetch Addresses read from the Register file
TP3_SP3_data	TP3→SP3	512	4 texture results
TPx_SPx_gpr_dst	TPx→SPx	7	Write address into the gprs
TPx_SPx_gpr_cmask	TPx→SPx	4	Channel mask

~~24.1.14~~25.1.17 *Sequencer to Fetch Unit bus*

Formatted: Bullets and Numbering

Once every clock, the fetch unit sends to the sequencer on which clause it is now working and if the data in the registers is ready or not. This way the sequencer can update the fetch counters for the reservation station fifos. The sequencer also provides the instruction and constants for the fetch to execute and the address in the register file where to write the fetch return data.

Name	**Direction**	**Bits**	**Description**
TPx_SQ_data_rdy	TPx→ SQ	1	Data ready
TPx_SQ_clause_num	TPx→ SQ	3	Clause number
SQ_TPx_const	SQ→TPx	64	Fetch state sent over 4 clocks
SQ_TPx_instuct	SQ→TPx	24	Fetch instruction sent over 4 clocks
SQ_TPx_end_of_clause	SQ→TPx	1	Last instruction of the clause

SQ_TPx_phase	SQ→TPx	2	Write phase signal
SQ_TP0_lod_correct	SQ→TP0	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP0_pmask	SQ→TP0	4	Pixel mask 1 bit per pixel
SQ_TP1_lod_correct	SQ→TP1	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP1_pmask	SQ→TP1	4	Pixel mask 1 bit per pixel
SQ_TP2_lod_correct	SQ→TP2	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP2_pmask	SQ→TP2	4	Pixel mask 1 bit per pixel
SQ_TP3_lod_correct	SQ→TP3	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP3_pmask	SQ→TP3	4	Pixel mask 1 bit per pixel
SQ_TPx_clause_num	SQ→TPx	3	Clause number
SQ_TPx_write_gpr_index	SQ->TPx	7	Index into Register file for write of returned Fetch Data

~~24.1.15~~25.1.18 *Sequencer to SP: GPR control*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SPx_gpr_wr_addr	SQ→SPx	7	Write address
SQ_SPx_gpr_rd_addr	SQ→SPx	7	Read address
SQ_SPx_gpr_re_addr	SQ→SPx	1	Read Enable
SQ_SPx_gpr_we_addr	SQ→SPx	1	Write Enable
SQ_SPx_gpr_phase_mux	SQ→SPx	2	The phase mux
SQ_SPx_gpr_channel_mask	SQ→SPx	4	The channel mask
SQ_SP0_gpr_pixel_mask	SQ→SP0	4	The pixel mask
SQ_SP1_gpr_pixel_mask	SQ→SP1	4	The pixel mask
SQ_SP2_gpr_pixel_mask	SQ→SP2	4	The pixel mask
SQ_SP3_gpr_pixel_mask	SQ→SP3	4	The pixel mask

~~24.1.16 *Sequencer to SPx: Parameter cache write control*~~

Formatted: Bullets and Numbering

~~24.1.17~~25.1.19 *Sequencer to SPx: Instructions*

Name	Direction	Bits	Description
SQ_SPx_instruct_start	SQ→SPx	1	Instruction start
SQ_SP_instruct	SQ→SPx	20	Instruction sent over 4 clocks
SQ_SPx_stall	SQ→SPx	1	Stall signal
SQ_SPx_Shader_Count	SQ→SPx	3	Each set of four pixels or vectors is exported over eight clocks. This field specifies where the SP is in that sequence.
SQ_SPx_Shader_Last	SQ→SPx	1	Asserted on the first shader count of the last export of the clause
SQ_SP0_Shader_PixelValid	SQ→SP0	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP0_Shader_WordValid	SQ→SP0	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors
SQ_SP1_Shader_PixelValid	SQ→SP1	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP1_Shader_WordValid	SQ→SP1	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors
SQ_SP2_Shader_PixelValid	SQ→SP2	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP2_Shader_WordValid	SQ→SP2	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16

			pixels or vectors
SQ_SP3_Shader_PixelValid	SQ→SP3	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP3_Shader_WordValid	SQ→SP3	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors

~~24.1.18~~25.1.20 *SP to Sequencer: Constant address load*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SP0_SQ_const_addr	SP0→SQ	36	Constant address load to the sequencer
SP0_SQ_valid	SP0→SQ	1	Data valid
SP1_SQ_const_addr	SP1→SQ	36	Constant address load to the sequencer
SP1_SQ_valid	SP1→SQ	1	Data valid
SP2_SQ_const_addr	SP2→SQ	36	Constant address load to the sequencer
SP2_SQ_valid	SP2→SQ	1	Data valid
SP3_SQ_const_addr	SP3→SQ	36	Constant address load to the sequencer
SP3_SQ_valid	SP3→SQ	1	Data valid

~~24.1.19~~25.1.21 *Sequencer to SPx: constant broadcast*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SPx_constant	SQ→SPx	128	Constant broadcast

~~24.1.20~~25.1.22 *SP0 to Sequencer: Kill vector load*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SP0_SQ_kill_vect	SP0→SQ	4	Kill vector load
SP1_SQ_kill_vect	SP1→SQ	4	Kill vector load
SP2_SQ_kill_vect	SP2→SQ	4	Kill vector load
SP3_SQ_kill_vect	SP3→SQ	4	Kill vector load

~~24.1.21~~25.1.23 *SQ to CP: RBBM bus*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_RBB_rs	SQ→CP	1	Read Strobe
SQ_RBB_rd	SQ→CP	32	Read Data
SQ_RBBM_nrtrtr	SQ→CP	1	Optional
SQ_RBBM_rtr	SQ→CP	1	Real-Time (Optional)

~~24.1.22~~25.1.24 *CP to SQ: RBBM bus*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
rbbm_we	CP→SQ	1	Write Enable
rbbm_a	CP→SQ	18	Address -- Upper Extent is TBD
rbbm_wd	CP→SQ	32	Data
rbbm_be	CP→SQ	4	Byte Enables
rbbm_re	CP→SQ	1	Read Enable
rbb_rs0	CP→SQ	1	Read Return Strobe 0
rbb_rs1	CP→SQ	1	Read Return Strobe 1
rbb_rd0	CP→SQ	32	Read Data 0
rbb_rd1	CP→SQ	32	Read Data 0
RBBM_SQ_soft_reset	CP→SQ	1	Soft Reset

Formatted: Bullets and Numbering

~~25.~~26. Examples of program executions

~~25.1.1~~26.1.1 *Sequencer Control of a Vector of Vertices*

1. PA sends a vector of 64 vertices (actually vertex indices – 32 bits/index for 2048 bit total) to the RE's Vertex FIFO
   - state pointer as well as tag into position cache is sent along with vertices
   - space was allocated in the position cache for transformed position before the vector was sent
   - **also before the vector is sent to the RE, the CP has loaded the global instruction store with the vertex shader program (using the MH?)**
   - The vertex program is assumed to be loaded when we receive the vertex vector.
     - the SEQ then accesses the IS base for this shader using the local state pointer (provided to all sequencers by the RBBM when the CP is done loading the program)
2. SEQ arbitrates between the Pixel FIFO and the Vertex FIFO – basically the Vertex FIFO always has priority
   - at this point the vector is removed from the Vertex FIFO
   - the arbiter is not going to select a vector to be transformed if the parameter cache is full unless the pipe as nothing else to do (ie no pixels are in the pixel fifo).
3. SEQ allocates space in the SP register file for index data plus GPRs used by the program
   - the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer that came down with the vertices
   - SEQ will not send vertex data until space in the register file has been allocated
4. SEQ sends the vector to the SP register file over the RE_SP interface (which has a bandwidth of 2048 bits/cycle)
   - the 64 vertex indices are sent to the 64 register files over 4 cycles
     - RF0 of SU0, SU1, SU2, and SU3 is written the first cycle
     - RF1 of SU0, SU1, SU2, and SU3 is written the second cycle
     - RF2 of SU0, SU1, SU2, and SU3 is written the third cycle
     - RF3 of SU0, SU1, SU2, and SU3 is written the fourth cycle
   - the index is written to the least significant 32 bits **(floating point format?) (what about compound indices)** of the 128-bit location within the register file (w); the remaining data bits are set to zero (x, y, z)
5. SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of fetch state machine 0, or TSM0 FIFO)
   - the control packet contains the state pointer, the tag to the position cache and a register file base pointer.
6. TSM0 accepts the control packet and fetches the instructions for fetch clause 0 from the global instruction store
   - TSM0 was first selected by the TSM arbiter before it could start
7. all instructions of fetch clause 0 are issued by TSM0
8. the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)
   - TSM0 does not wait for requests made to the Fetch Unit to complete; it passes the register file write index for the fetch data to the TU, which will write the data to the RF as it is received
   - once the TU has written all the data to the register files, it increments a counter that is associated with ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go ahead start to execute the ALU clause
9. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store
10. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of fetch state machine 1, or TSM1 FIFO)
11. the control packet continues to travel down the path of reservation stations until all clauses have been executed
    - position can be exported in ALU clause 3 (or 4?); the data (and the tag) is sent over a position bus (which is shared with all four shader pipes) back to the PA's position cache
    - A parameter cache pointer is also sent along with the position data. This tells to the PA where the data is going to be in the parameter cache.
      - there is a position export FIFO in the SP that buffers position data before it gets sent back to the PA

    - the ASM arbiter will prevent a packet from starting an exporting clause if the position export FIFO is full
- parameter data is exported in clause 7 (as well as position data if it was not exported earlier)
    - parameter data is sent to the Parameter Cache over a dedicated bus
    - the SEQ allocates storage in the Parameter Cache, and the SEQ deallocates that space when there is no longer a need for the parameters (it is told by the PA when using a token).
    - the ASM arbiter will prevent a packet from starting on ASM7 if the parameter cache (or the position buffer if position is being exported) is full

12. after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program

~~25.1.2~~26.1.2 *Sequencer Control of a Vector of Pixels*

Formatted: Bullets and Numbering

1. **As with vertex shader programs, pixel shaders are loaded into the global instruction store by the CP**
    - At this point it is assumed that the pixel program is loaded into the instruction store and thus ready to be read.
2. the RE's Pixel FIFO is loaded with the barycentric coordinates for pixel quads by the detailed walker
    - the state pointer and the LOD correction bits are also placed in the Pixel FIF0
    - the Pixel FIFO is wide enough to source four quad's worth of barycentrics per cycle
3. SEQ arbitrates between Pixel FIFO and Vertex FIFO – when there are no vertices pending OR there is no space left in the register files for vertices, the Pixel FIFO is selected
4. SEQ allocates space in the SP register file for all the GPRs used by the program
    - the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer
    - SEQ will not allow interpolated data to be sent to the shader until space in the register file has been allocated
5. SEQ controls the transfer of interpolated data to the SP register file over the RE_SP interface (which has a bandwidth of 2048 bits/cycle). See interpolated data bus diagrams for details.
6. SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of fetch state machine 0, or TSM0 FIFO)
    - note that there is a separate set of reservation stations/arbiters/state machines for vertices and for pixels
    - the control packet contains the state pointer, the register file base pointer, and the LOD correction bits
    - all other information (such as quad address for example) travels in a separate FIFO
7. TSM0 accepts the control packet and fetches the instructions for fetch clause 0 from the global instruction store
    - TSM0 was first selected by the TSM arbiter before it could start
8. all instructions of fetch clause 0 are issued by TSM0
9. the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)
    - TSM0 does not wait for fetch requests made to the Fetch Unit to complete; it passes the register file write index for the fetch data to the TU, which will write the data to the RF as it is received
    - once the TU has written all the data for a particular clause to the register files, it increments a counter that is associated with the ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go ahead and pop the FIFO and start to execute the ALU clause
10. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store
11. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of fetch state machine 1, or TSM1 FIFO)
12. the control packet continues to travel down the path of reservation stations until all clauses have been executed
    - pixel data is exported in the last ALU clause (clause 7)
        - it is sent to an output FIFO where it will be picked up by the render backend
        - the ASM arbiter will prevent a packet from starting on ASM7 if the output FIFO is full
13. after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program

~~25.1.3~~26.1.3 *Notes*

Formatted: Bullets and Numbering

14. The state machines and arbiters will operate ahead of time so that they will be able to immediately start the real threads or stall.

15. The register file base pointer for a vector needs to travel with the vector through the reservation stations, but the instruction store base pointer does not – this is because the RF pointer is different for all threads, but the IS pointer is only different for each state and thus can be accessed via the state pointer

16. Waterfalling still needs to be specked out.

~~26.~~27. Open issues

Formatted: Bullets and Numbering

There is currently an issue with constants. If the constants are not the same for the whole vector of vertices, we don't have the bandwidth from the fetch store to feed the ALUs. Two solutions exists for this problem:

1) Let the compiler handle the case and put those instructions in a fetch clause so we can use the bandwidth there to operate. This requires a significant amount of temporary storage in the register store.
2) Waterfall down the pipe allowing only at a given time the vertices having the same constants to operate in parallel. This might in the worst case slow us down by a factor of 16.

Need to do some testing on the size of the register file as well as on the register file allocation method (dynamic VS static).

Saving power?

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 2015~~11~~ December, 20016	GEN-CXXXXX-REVA	1 of 48

Author: Laurent Lefebvre

Issue To:	**Copy No:**

R400 Sequencer Specification

SQ

Version 1.54

Overview: This is an architectural specification for the R400 Sequencer block (SEQ). It provides an overview of the required capabilities and expected uses of the block. It also describes the block interfaces, internal sub-blocks, and provides internal state diagrams.

AUTOMATICALLY UPDATED FIELDS:

Document Location: C:\perforce\r400\doc_lib\design\blocks\sq\R400_Sequencer.doc

Current Intranet Search Title: R400 Sequencer Specification

APPROVALS	
Name/Dept	Signature/Date

Remarks:

THIS DOCUMENT CONTAINS CONFIDENTIAL INFORMATION THAT COULD BE SUBSTANTIALLY DETRIMENTAL TO THE INTEREST OF ATI TECHNOLOGIES INC. THROUGH UNAUTHORIZED USE OR DISCLOSURE.

Table Of Contents

1. OVERVIEW **8~~6~~**
1.1 Top Level Block Diagram 10~~8~~
1.2 Data Flow graph 12~~10~~
1.3 Control Graph 13~~11~~
2. INTERPOLATED DATA BUS **13~~11~~**
3. INSTRUCTION STORE **16~~14~~**
4. SEQUENCER INSTRUCTIONS **18~~16~~**
5. CONSTANT STORES **18~~16~~**
5.1 Memory organizations 18~~16~~
5.2 Management of the re-mapping tables 18~~16~~
5.2.1 Dirty bits 20~~18~~
5.2.2 Free List Block 20~~18~~
5.2.3 De-allocate Block 20~~18~~
5.2.4 Operation of Incremental model 20~~18~~
5.3 Constant Store Indexing 21~~19~~
5.4 Real Time Commands 21~~19~~
6. LOOPING AND BRANCHES **22~~20~~**
6.1 The controlling state. 22~~20~~
6.2 The Control Flow Program 22~~20~~
6.3 Data dependant predicate instructions 25~~23~~
6.4 HW Detection of PV,PS 25~~23~~
6.5 Register file indexing 26~~23~~
6.6 Predicated Instruction support for Texture clauses 26~~24~~
6.7 Debugging the Shaders 26~~24~~
6.7.1 Method 1: Debugging registers 26~~24~~
6.7.2 Method 2: Exporting the values in the GPRs (12) 27~~25~~
7. PIXEL KILL MASK **27~~25~~**
8. MULTIPASS VERTEX SHADERS (HOS) **27~~25~~**
9. REGISTER FILE ALLOCATION **27~~25~~**
10. FETCH ARBITRATION **28~~26~~**
11. ALU ARBITRATION **28~~26~~**
12. HANDLING STALLS **29~~27~~**
13. CONTENT OF THE RESERVATION STATION FIFOS **29~~27~~**
14. THE OUTPUT FILE **29~~27~~**
15. IJ FORMAT **29~~27~~**
15.1 Interpolation of constant attributes 30~~28~~
16. THE PARAMETER CACHE **30~~28~~**
17. VERTEX POSITION EXPORTING **30~~28~~**
18. EXPORTING ARBITRATION **31~~28~~**
19. EXPORT TYPES **31~~29~~**
19.1 Vertex Shading 31~~29~~
19.2 Pixel Shading 31~~29~~
20. SPECIAL INTERPOLATION MODES **31~~29~~**

20.1 Real time commands .... 31~~29~~
20.2 Sprites/ XY screen coordinates/ FB information .... 32~~30~~
20.3 Auto generated counters .... 32~~30~~
20.3.1 Vertex shaders .... 32~~30~~
20.3.2 Pixel shaders .... 32~~30~~
21. STATE MANAGEMENT .... 33~~31~~
21.1 Parameter cache synchronization .... 33~~31~~
22. XY ADDRESS IMPORTS .... 33~~31~~
22.1 Vertex indexes imports .... 33~~31~~
23. REGISTERS .... 33~~31~~
23.1 Control .... 33~~31~~
23.2 Context .... 34~~32~~
24. DEBUG REGISTERS .... 35~~33~~
24.1 Context .... 35~~33~~
25. INTERFACES .... 35~~33~~
25.1 External Interfaces .... 35~~33~~
25.1.1 SC to SQ : IJ Control bus .... 35~~33~~
25.1.2 SQ to SP: Interpolator bus .... 36~~34~~
25.1.3 SQ to SP: GPR Input Mux select .... 36~~34~~
25.1.4 SQ to SP: Parameter Cache Read control bus .... 37~~35~~
25.1.5 SQ to SX: Parameter Cache Mux control Bus .... 37~~35~~
25.1.6 SQ to SP: Staging Register Data .... 37~~35~~
25.1.7 PA to SQ : Vertex interface .... 37~~35~~
25.1.8 SQ to CP: State report .... 41~~39~~
25.1.9 SQ to SX: Control bus .... 41~~39~~
25.1.10 SX to SQ : Output file control .... 41~~39~~
25.1.11 SQ to TP: Control bus .... 41~~39~~
25.1.12 TP to SQ: Texture stall .... 42~~40~~
25.1.13 SQ to SP: Texture stall .... 42~~40~~
25.1.14 SQ to SP: GPR and Parameter cache control .... 42~~40~~
25.1.15 SQ to SPx: Instructions .... 43~~41~~
25.1.16 SP to SQ: Constant address load .... 44~~41~~
25.1.17 SQ to SPx: constant broadcast .... 45~~41~~
25.1.18 SP0 to SQ: Kill vector load .... 45~~42~~
25.1.19 SQ to CP: RBBM bus .... 45~~42~~
25.1.20 CP to SQ: RBBM bus .... 45~~42~~
26. EXAMPLES OF PROGRAM EXECUTIONS .... 45~~42~~
26.1.1 Sequencer Control of a Vector of Vertices .... 45~~42~~
26.1.2 Sequencer Control of a Vector of Pixels .... 46~~43~~
26.1.3 Notes .... 47~~44~~

27. OPEN ISSUES 4744
~~1. OVERVIEW .... 5~~
~~1.1 Top Level Block Diagram .... 7~~
~~1.2 Data Flow graph .... 9~~
~~1.3 Control Graph .... 10~~
~~2. INTERPOLATED DATA BUS .... 10~~
~~3. INSTRUCTION STORE .... 13~~
~~4. SEQUENCER INSTRUCTIONS .... 15~~
~~5. CONSTANT STORES .... 15~~
~~5.1 Memory organizations .... 15~~
~~5.2 Management of the re-mapping tables .... 15~~
~~5.2.1 Dirty bits .... 17~~
~~5.2.2 Free List Block .... 17~~
~~5.2.3 De-allocate Block .... 17~~
~~5.2.4 Operation of Incremental model .... 17~~
~~5.3 Constant Store Indexing .... 18~~
~~5.4 Real Time Commands .... 18~~
~~6. LOOPING AND BRANCHES .... 19~~
~~6.1 The controlling state .... 19~~
~~6.2 The Control Flow Program .... 19~~
~~6.3 Data dependant predicate instructions .... 22~~
~~6.4 HW Detection of PV,PS .... 22~~
~~6.5 Register file indexing .... 22~~
~~6.6 Predicated Instruction support for Texture clauses .... 23~~
~~6.7 Debugging the Shaders .... 23~~
~~6.7.1 Method 1: Debugging registers .... 23~~
~~6.7.2 Method 2: Exporting the values in the GPRs (12) .... 24~~
~~7. PIXEL KILL MASK .... 24~~
~~8. MULTIPASS VERTEX SHADERS (HOS) .... 24~~
~~9. REGISTER FILE ALLOCATION .... 24~~
~~10. FETCH ARBITRATION .... 25~~
~~11. ALU ARBITRATION .... 25~~
~~12. HANDLING STALLS .... 26~~
~~13. CONTENT OF THE RESERVATION STATION FIFOS .... 26~~
~~14. THE OUTPUT FILE .... 26~~
~~15. IJ FORMAT .... 26~~
~~15.1 Interpolation of constant attributes .... 27~~
~~16. THE PARAMETER CACHE .... 27~~
~~17. VERTEX POSITION EXPORTING .... 27~~
~~18. EXPORTING ARBITRATION .... 27~~
~~19. EXPORT TYPES .... 28~~
~~19.1 Vertex Shading .... 28~~
~~19.2 Pixel Shading .... 28~~
~~20. SPECIAL INTERPOLATION MODES .... 28~~
~~20.1 Real time commands .... 28~~
~~20.2 Sprites/ XY screen coordinates/ FB information .... 29~~
~~20.3 Auto generated counters .... 29~~
~~20.3.1 Vertex shaders .... 29~~
~~20.3.2 Pixel shaders .... 29~~
~~21. STATE MANAGEMENT .... 30~~
~~21.1 Parameter cache synchronization .... 30~~
~~22. XY ADDRESS IMPORTS .... 30~~
~~22.1 Vertex indexes imports .... 30~~
~~23. REGISTERS .... 30~~
~~23.1 Control .... 30~~
~~23.2 Context .... 31~~
~~24. DEBUG REGISTERS .... 32~~
~~24.1 Context .... 32~~

~~25. INTERFACES 32~~
~~25.1 External Interfaces 32~~
~~25.1.1 SC to SQ : IJ Control bus 32~~
~~25.1.2 SQ to SP: Interpolator bus 33~~
~~25.1.3 SQ to SP: GPR Input Mux select 33~~
~~25.1.4 SQ to SP: Parameter Cache Read control bus 34~~
~~25.1.5 SQ to SX: Parameter Cache Mux control Bus 34~~
~~25.1.6 SQ to SP: Staging Register Data 34~~
~~25.1.7 PA to SQ : Vertex interface 34~~
~~25.1.8 SQ to CP: State report 38~~
~~25.1.9 SQ to SX: Control bus 38~~
~~25.1.10 SX to SQ : Output file control 38~~
~~25.1.11 SQ to TP: Control bus 38~~
~~25.1.12 TP to SQ: Texture stall 39~~
~~25.1.13 SQ to SP: Texture stall 39~~
~~25.1.14 SQ to SP: GPR and Parameter cache control 39~~
~~25.1.15 SQ to SPx: Instructions 40~~
~~25.1.16 SP to SQ: Constant address load 40~~
~~25.1.17 SQ to SPx: constant broadcast 40~~
~~25.1.18 SP0 to SQ: Kill vector load 41~~
~~25.1.19 SQ to CP: RBBM bus 41~~
~~25.1.20 CP to SQ: RBBM bus 41~~
~~26. EXAMPLES OF PROGRAM EXECUTIONS 41~~
~~26.1.1 Sequencer Control of a Vector of Vertices 41~~
~~26.1.2 Sequencer Control of a Vector of Pixels 42~~
~~26.1.3 Notes 43~~
~~27. OPEN ISSUES 43~~
~~1. OVERVIEW 4~~
~~1.1 Top Level Block Diagram 6~~
~~1.2 Data Flow graph 8~~
~~1.3 Control Graph 9~~
~~2. INTERPOLATED DATA BUS 9~~
~~3. INSTRUCTION STORE 12~~
~~4. SEQUENCER INSTRUCTIONS 14~~
~~5. CONSTANT STORES 14~~
~~5.1 Memory organizations 14~~
~~5.2 Management of the remaping tables 14~~
~~5.3 Constant Store Indexing 14~~
~~5.4 14~~
~~5.5 Real Time Commands 15~~
~~6. LOOPING AND BRANCHES 15~~
~~6.1 The controlling state 15~~
~~6.2 The Control Flow Program 15~~
~~6.3 Data dependant predicate instructions 18~~
~~6.4 HW Detection of PV,PS 18~~
~~6.5 Register file indexing 18~~
~~6.6 Predicated Instruction support for Texture clauses 19~~
~~6.7 Debugging the Shaders 19~~
~~6.7.1 Method 1: Debugging registers 19~~
~~6.7.2 Method 2: Exporting the values in the GPRs (12) 20~~
~~7. PIXEL KILL MASK 20~~
~~8. MULTIPASS VERTEX SHADERS (HOS) 20~~
~~9. REGISTER FILE ALLOCATION 20~~
~~10. FETCH ARBITRATION 21~~
~~11. ALU ARBITRATION 21~~
~~12. HANDLING STALLS 22~~
~~13. CONTENT OF THE RESERVATION STATION FIFOS 22~~
~~14. THE OUTPUT FILE 22~~

~~15. IJ FORMAT.......22~~
~~15.1 Interpolation of constant attributes.......23~~
~~16. THE PARAMETER CACHE.......23~~
~~17. VERTEX POSITION EXPORTING.......23~~
~~18. EXPORTING ARBITRATION.......23~~
~~19. EXPORT TYPES.......24~~
~~19.1 Vertex Shading.......24~~
~~19.2 Pixel Shading.......24~~
~~20. SPECIAL INTERPOLATION MODES.......24~~
~~20.1 Real time commands.......24~~
~~20.2 Sprites/ XY screen coordinates/ FB information.......25~~
~~20.3 Auto generated counters.......25~~
~~20.3.1 Vertex shaders.......25~~
~~20.3.2 Pixel shaders.......25~~
~~21. STATE MANAGEMENT.......26~~
~~22. XY ADDRESS IMPORTS.......26~~
~~22.1 Vertex indexes imports.......26~~
~~23. REGISTERS.......26~~
~~23.1 Control.......26~~
~~23.2 Context.......27~~
~~24. DEBUG REGISTERS.......28~~
~~24.1 Context.......28~~
~~25. INTERFACES.......28~~
~~25.1 External Interfaces.......28~~
~~25.1.1 SC to SP : IJ bus.......28~~
~~25.1.2 SC to SQ : IJ Control bus.......28~~
~~25.1.3 SQ to SC: Vertex/Pixel shader synchronization bus.......29~~
~~25.1.4 SQ to SP: Interpolator bus.......29~~
~~25.1.5 SQ to SP: GPR Input Mux select.......29~~
~~25.1.6 SQ to SPx: Parameter cache write control.......30~~
~~25.1.7 SQ to SP: Parameter Cache Read control bus.......30~~
~~25.1.8 SQ to SX: Parameter Cache Mux control Bus.......30~~
~~25.1.9 SP to SX: Parameter data.......30~~
~~25.1.10 SX to Interpolators: Parameter Cache Return bus.......30~~
~~25.1.11 SQ to SP0: Staging Register Data.......31~~
~~25.1.12 PA to SQ : Vertex interface.......31~~
~~25.1.13 SQ to CP: State report.......34~~
~~25.1.14 SP to SX : Pixel/Vertex write to SX.......34~~
~~25.1.15 SQ to SX: Control bus.......34~~
~~25.1.16 SX to SQ : Output file control.......35~~
~~25.1.17 Shader Engine to Fetch Unit Bus.......35~~
~~25.1.18 Sequencer to Fetch Unit bus.......35~~
~~25.1.19 Sequencer to SP: GPR control.......36~~
~~25.1.20 Sequencer to SPx: Instructions.......36~~
~~25.1.21 SP to Sequencer: Constant address load.......37~~
~~25.1.22 Sequencer to SPx: constant broadcast.......37~~
~~25.1.23 SP0 to Sequencer: Kill vector load.......37~~
~~25.1.24 SQ to CP: RBBM bus.......37~~
~~25.1.25 CP to SQ: RBBM bus.......37~~
~~26. EXAMPLES OF PROGRAM EXECUTIONS.......37~~
~~26.1.1 Sequencer Control of a Vector of Vertices.......37~~
~~26.1.2 Sequencer Control of a Vector of Pixels.......38~~
~~26.1.3 Notes.......39~~
~~27. OPEN ISSUES.......40~~

Revision Changes:

Rev 0.1 (Laurent Lefebvre) Date: May 7, 2001	First draft.
Rev 0.2 (Laurent Lefebvre) Date : July 9, 2001	Changed the interfaces to reflect the changes in the SP. Added some details in the arbitration section.
Rev 0.3 (Laurent Lefebvre) Date : August 6, 2001	Reviewed the Sequencer spec after the meeting on August 3, 2001.
Rev 0.4 (Laurent Lefebvre) Date : August 24, 2001	Added the dynamic allocation method for register file and an example (written in part by Vic) of the flow of pixels/vertices in the sequencer.
Rev 0.5 (Laurent Lefebvre) Date : September 7, 2001	Added timing diagrams (Vic)
Rev 0.6 (Laurent Lefebvre) Date : September 24, 2001	Changed the spec to reflect the new R400 architecture. Added interfaces.
Rev 0.7 (Laurent Lefebvre) Date : October 5, 2001	Added constant store management, instruction store management, control flow management and data dependant predication.
Rev 0.8 (Laurent Lefebvre) Date : October 8, 2001	Changed the control flow method to be more flexible. Also updated the external interfaces.
Rev 0.9 (Laurent Lefebvre) Date : October 17, 2001	Incorporated changes made in the 10/18/01 control flow meeting. Added a NOP instruction, removed the conditional_execute_or_jump. Added debug ~~registers~~registers.
Rev 1.0 (Laurent Lefebvre) Date : October 19, 2001	Refined interfaces to RB. Added state ~~registers~~registers.
Rev 1.1 (Laurent Lefebvre) Date : October 26, 2001	Added SEQ→SP0 interfaces. Changed delta precision. Changed VGT→SP0 interface. Debug Methods added.
Rev 1.2 (Laurent Lefebvre) Date : November 16, 2001	Interfaces greatly refined. Cleaned up the spec.
Rev 1.3 (Laurent Lefebvre) Date : November 26, 2001	Added the different interpolation modes.
Rev 1.4 (Laurent Lefebvre) Date : December 6, 2001	Added the auto incrementing counters. Changed the VGT→SQ interface. Added content on constant management. Updated ~~registers~~GPRs.
Rev 1.5 (Laurent Lefebvre) Date : December 11, 2001	Removed from the spec all interfaces that weren't directly tied to the SQ. Added explanations on constant management. Added PA→SQ synchronization fields and explanation.

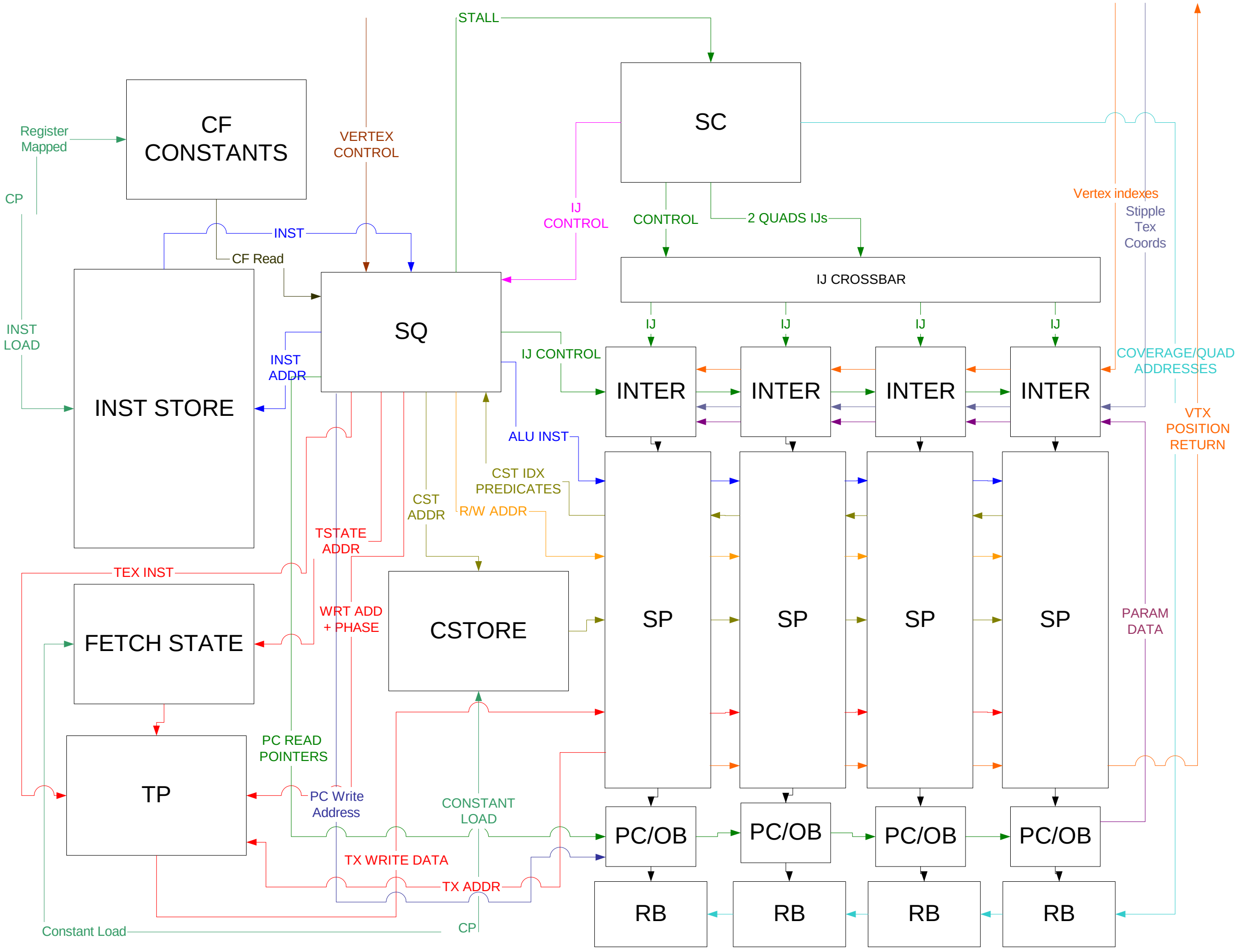
1. Overview

The sequencer is based on the R300 design. It chooses two ALU clauses and a fetch clause to execute, and executes all of the instructions in a clause before looking for a new clause of the same type. Two ALU clauses are executed interleaved to hide the ALU latency. Each vector will have eight fetch and eight ALU clauses, but clauses do not need to contain instructions. A vector of pixels or vertices ping-pongs along the sequencer FIFO, bouncing from fetch reservation station to alu reservation station. A FIFO exists between each reservation stage, holding up vectors until the vector currently occupying a reservation station has left. A vector at a reservation station can be chosen to execute. The sequencer looks at all eight alu reservation stations to choose an alu clause to execute and all eight fetch stations to choose a fetch clause to execute. The arbitrator will give priority to clauses/reservation stations closer to the bottom of the pipeline. It will not execute an alu clause until the fetch fetches initiated by the previous fetch clause have completed. There are two separate sets of reservation stations, one for pixel vectors and one for vertices vectors. This way a pixel can pass a vertex and a vertex can pass a pixel.

To support the shader pipe the sequencer also contains the shader instruction cache, constant store, control flow constants and texture state. The four shader pipes also execute the same instruction thus there is only one sequencer for the whole chip.

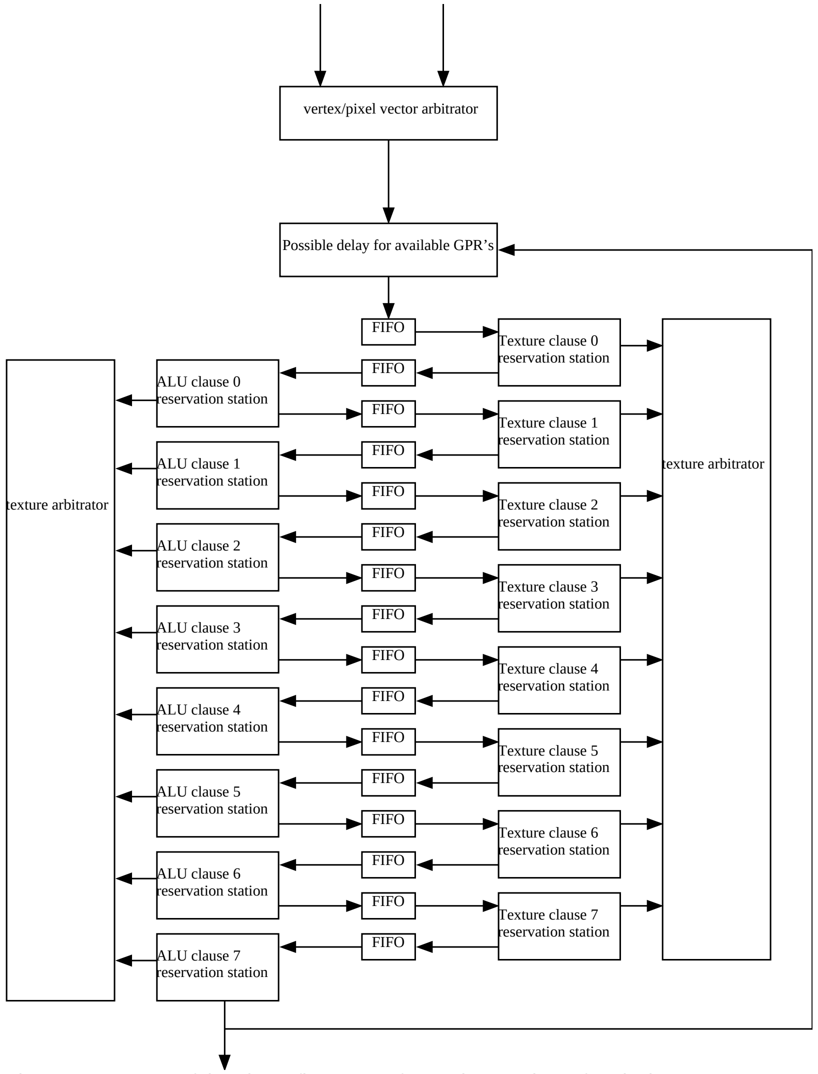
The sequencer first arbitrates between vectors of 64 vertices that arrive directly from primitive assembly and vectors of 16 quads (64 pixels) that are generated in the scan converter.

The vertex or pixel program specifies how many GPRs it needs to execute. The sequencer will not start the next vector until the needed space is available in the GPRs.



STALL
VERTEX
CONTROL
Register
Mapped
CF
CONSTANTS
CP
INST
CF Read
SC
IJ
CONTROL
CONTROL
2 QUADS IJs
Vertex indexes
Stipple
Tex
Coords
IJ CROSSBAR
INST
LOAD
INST STORE
SQ
INST
ADDR
IJ CONTROL
IJ
IJ
IJ
IJ
INTER
INTER
INTER
INTER
COVERAGE/QUAD
ADDRESSES
VTX
POSITION
RETURN
ALU INST
CST IDX
PREDICATES
CST
ADDR
R/W ADDR
TSTATE
ADDR
TEX INST
FETCH STATE
WRT ADD
+ PHASE
CSTORE
SP
SP
SP
SP
PARAM
DATA
TP
PC READ
POINTERS
PC Write
Address
CONSTANT
LOAD
TX WRITE DATA
TX ADDR
PC/OB
PC/OB
PC/OB
PC/OB
RB
RB
RB
RB
Constant Load
CP

1.1 Top Level Block Diagram



vertex/pixel vector arbitrator
Possible delay for available GPR's
FIFO
Texture clause 0 reservation station
Texture clause 1 reservation station
Texture clause 2 reservation station
Texture clause 3 reservation station
Texture clause 4 reservation station
Texture clause 5 reservation station
Texture clause 6 reservation station
Texture clause 7 reservation station
ALU clause 0 reservation station
ALU clause 1 reservation station
ALU clause 2 reservation station
ALU clause 3 reservation station
ALU clause 4 reservation station
ALU clause 5 reservation station
ALU clause 6 reservation station
ALU clause 7 reservation station
texture arbitrator
texture arbitrator

There are two sets of the above figure, one for vertices and one for pixels.

Depending on the arbitration state, the sequencer will either choose a vertex or a pixel packet. The control packet consists of 3 bits of state, 7 bits for the base address of the Shader program and some information on the coverage to determine fetch LOD plus other various small state bits.

On receipt of a packet, the input state machine (not pictured but just before the first FIFO) allocated enough space in the ~~registers~~GPRs to store the interpolated values and temporaries. Following this, the barycentric coordinates (and

XY screen position if needed) are sent to the interpolator ~~buffers~~ which ~~are going to~~will use ~~these barycentric coordinates~~them to interpolate the parameters and place the ~~interpolated value~~results into the GPRs. Then, the input state machine stacks the packet in the first FIFO.

On receipt of a command, the level 0 fetch machine issues a ~~texture~~ fetch request to the TP and corresponding ~~register~~ GPR address for the fetch address (ta). A small command (tcmd) is passed to the fetch system identifying the current level number (0) as well as the ~~register~~ GPR write address for the fetch return data. One fetch request is sent every 4 clocks causing the texturing of sixteen 2x2s worth of data (or 64 vertices). Once all the requests are sent the packet is put in FIFO 1.

Upon receipt of the return data, the fetch unit writes the data to the register file using the write address that was provided by the level 0 fetch machine and sends the clause number (0) to the level 0 fetch state machine to signify that the write is done and thus the data is ready. Then, the level 0 fetch machine increments the counter of FIFO 1 to signify to the ALU ~~1~~ 0 that the data is ready to be processed.

On receipt of a command, the level 0 ALU machine first decrements the input FIFO 1 counter and then issues a complete set of level 0 shader instructions. For each instruction, the ALU state machine generates 3 source addresses, one destination address ~~(3 cycles later)~~ and an instruction. Once the last instruction has been issued, the packet is put into FIFO 2.

There will always be two active ALU clauses at any given time (and two arbiters). One arbiter will arbitrate over the odd instructions (4 clocks cycles) and the other one will arbitrate over the even instructions (4 clocks cycles). The only constraints between the two arbiters is that they are not allowed to pick the same clause number as the other one is currently working on if the packet is not of the same type (render state).

If the packet is a vertex packet, upon reaching ALU clause 3, it can export the position if the position is ready. So the arbiter must prevent ALU clause 3 to be selected if the positional buffer is full (or can't be accessed). Along with the positional data, if needed the sprite size and/or edge flags can also be sent.

{ISSUE: How do we handle parameter cache pointers (computed, semi-computed or not computed)?}

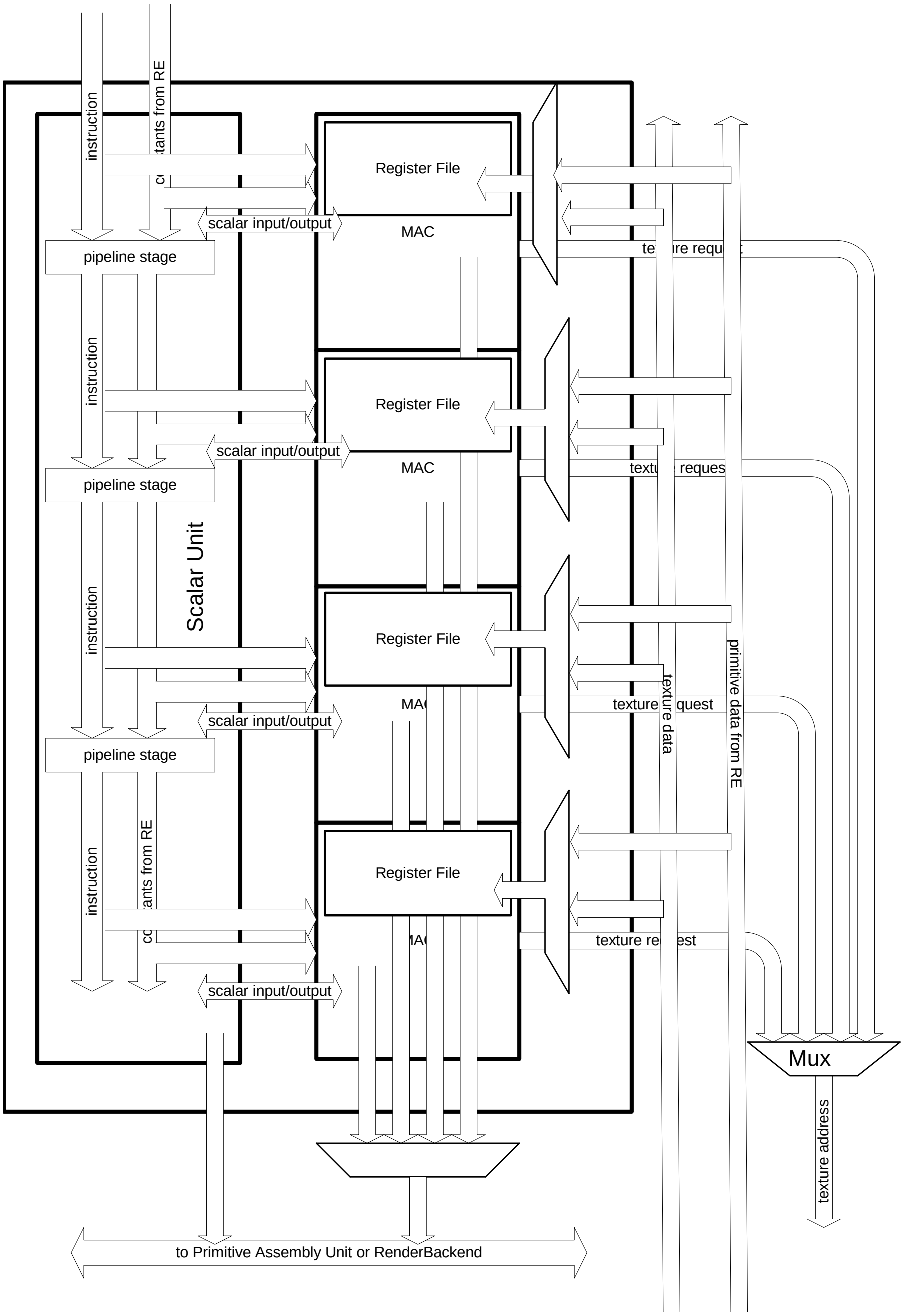
A special case is for multipass vertex shaders, which can export 12 parameters per last 6 clauses to the output buffer. If the output buffer is full or doesn't have enough space the sequencer will prevent such a vertex group to enter an exporting clause.

Multipass pixel shaders can export 12 parameters to memory from the last clause only (7).

All other clauses~~level~~ process in the same way until the packet finally reaches the last ALU machine (7).

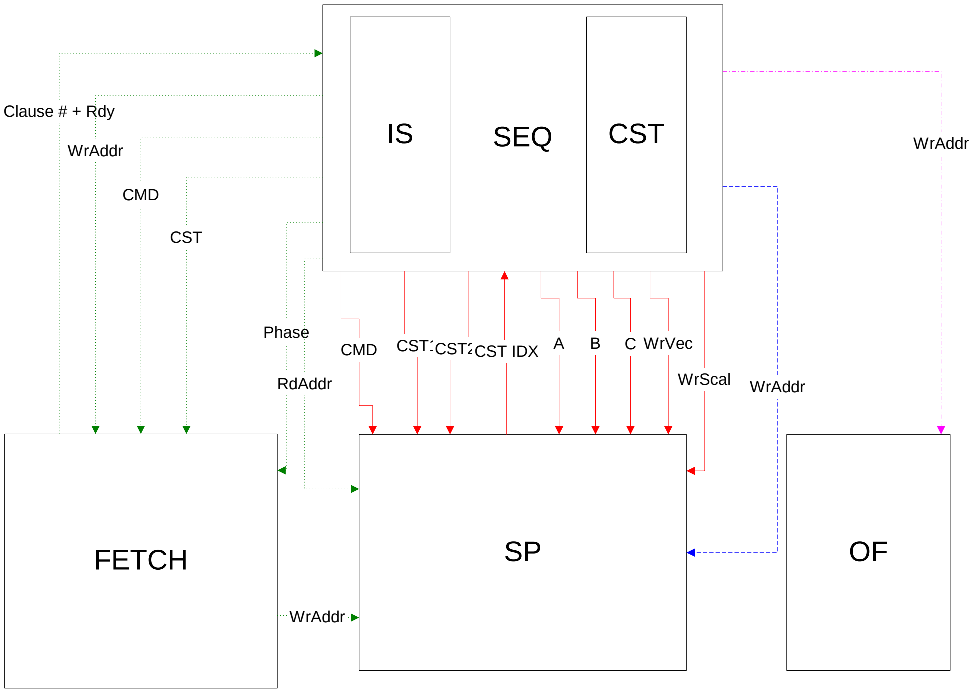
Only ~~two~~ one pair of interleaved ALU state machines may have access to the register file address bus or the instruction decode bus at one time. Similarly, only one fetch state machine may have access to the register file address bus at one time. Arbitration is performed by three arbiter blocks (two for the ALU state machines and one for the fetch state machines). The arbiters always favor the higher number state machines, preventing a bunch of half finished jobs from clogging up the register files.

1.2 Data Flow graph



The gray area represents blocks that are replicated 4 times per shader pipe (16 times on the overall chip).

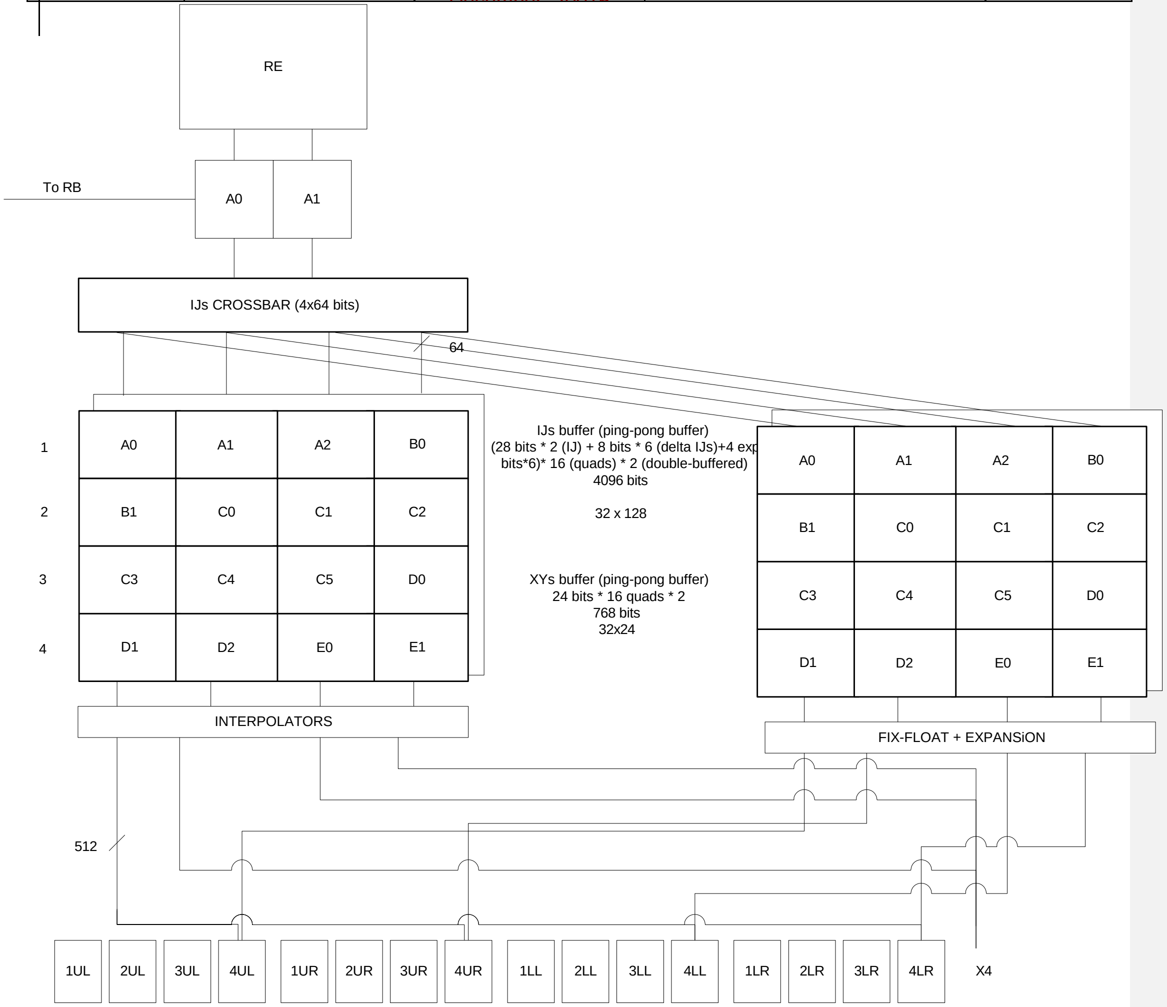
1.3 Control Graph



In green is represented the Fetch control interface, in red the ALU control interface, in blue the Interpolated/Vector control interface and in purple is the output file control interface.

2. Interpolated data bus

The interpolators contain an IJ buffer to pack the information as much as possible before writing it to the register file.


RE
To RB
A0
A1
IJs CROSSBAR (4x64 bits)
64
1
2
3
4
A0
A1
A2
B0
B1
C0
C1
C2
C3
C4
C5
D0
D1
D2
E0
E1
IJs buffer (ping-pong buffer)
(28 bits * 2 (IJ) + 8 bits * 6 (delta IJs)+4 exp
bits*6)* 16 (quads) * 2 (double-buffered)
4096 bits
32 x 128
XYs buffer (ping-pong buffer)
24 bits * 16 quads * 2
768 bits
32x24
A0
A1
A2
B0
B1
C0
C1
C2
C3
C4
C5
D0
D1
D2
E0
E1
INTERPOLATORS
FIX-FLOAT + EXPANSiON
512
1UL
2UL
3UL
4UL
1UR
2UR
3UR
4UR
1LL
2LL
3LL
4LL
1LR
2LR
3LR
4LR
X4

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21	T22	T23
											WRITES													
SP 0	A0	A0	XY A0	B1	B1	XY B1	C3	C3	XY C3				D1	D1	XY D1									
SP 1	A1	A1	XY A1				C0	C0	XY C0	C4	C4	XY C4	D2	D2	XY D2									
SP 2	A2	A2	XY A2				C1	C1	XY C1	C5	C5	XY C5				E0	E0	XY E0						
SP 3				B0	B0	XY B0	C2	C2	XY C2				D0	D0	XY D0	E1	E1	XY E1						
											READS													
SP 0	XY 0-3	XY 16-19	XY 32-35	XY 48-51	A0	B1	C3	D1					A0	B1	C3	D1					V 0-3	V 16-19	V 32-35	V 48-51
SP 1	XY 4-7	XY 20-23	XY 36-39	XY 52-55	A1		C4	D2		C0			A1		C4	D2		C0			V 4-7	V 20-23	V 36-39	V 52-55
SP 2	XY 8-11	XY 24-27	XY 40-43	XY 56-59	A2		C5			C1		E0	A2		C5			C1		E0	V 8-11	V 24-27	V 40-43	V 56-59
SP 3	XY 12-15	XY 28-31	XY 44-47	XY 60-63					B0	C2	D0	E1					B0	C2	D0	E1	V 12-15	V 28-31	V 44-47	V 60-63
	XY				P1								P2								VTX			

Above is an example of a tile ~~we~~ the sequencer might receive from the SC. The write side is how the data get stacked into the XY and IJ buffers, the read side is how the data is passed to the GPRs. The IJ information is packed in the IJ buffer ~~2~~4 quads at a time or two clocks. The sequencer allows at any given time as many as four quads to interpolate a parameter. They all have to come from the same primitive. Then the sequencer controls the write mask to the ~~register~~ GPRs to write the valid data in.

3. Instruction Store

There is going to be only one instruction store for the whole chip. It will contain 4096 instructions of 96 bits each.

It is likely to be a 1 port memory; we use 1 clock to load the ALU instruction, 1 clocks to load the Fetch instruction, 1 clock to load 2 control flow instructions and 1 clock to write instructions.

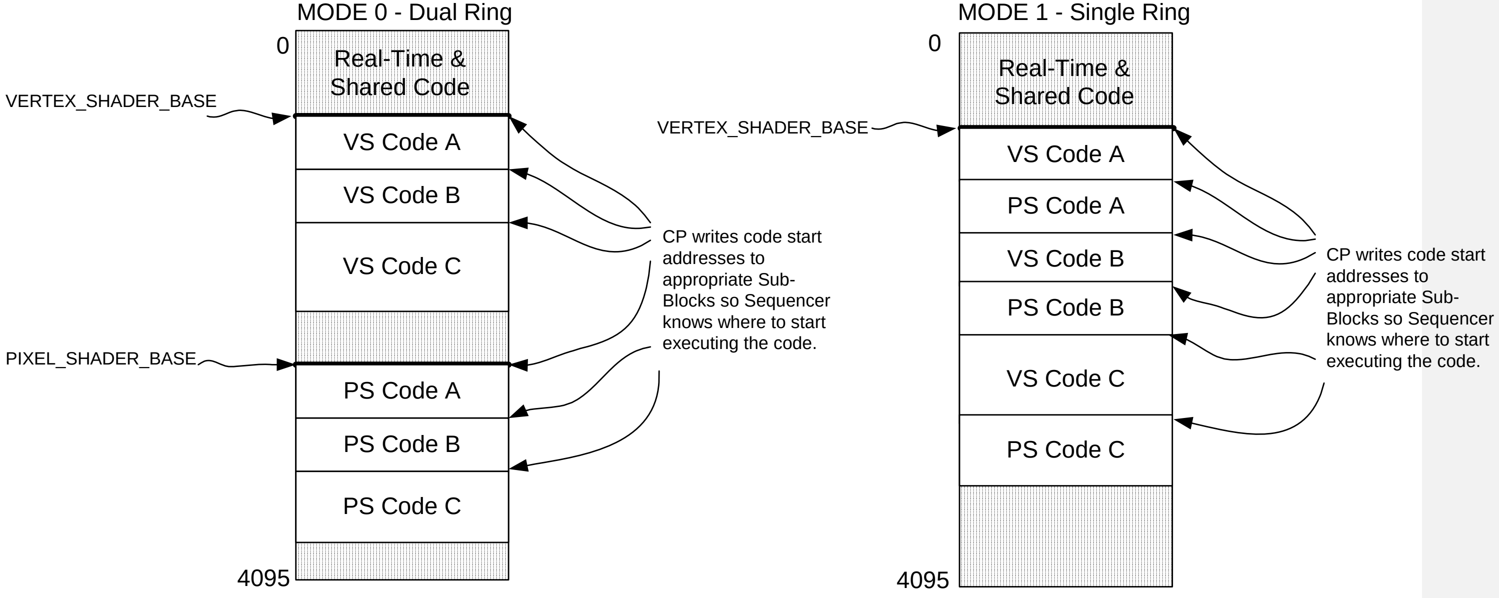
The instruction store is loaded by the CP thru the INST_DATA, INST_INDEX_PORT control ~~registers~~register. The INST_INDEX_PORT is auto-incremented on both reads and writes to the INST_DATA register.

The next picture shows the various modes the CP can load the memory. The Sequencer has to keep track of the loading modes in order to wrap around the correct boundaries. The MSB of the INST_INDEX_PORT register contains the packet type for the sequencer to know where it must wrap around. The wrap around points are arbitrary and they are specified in the VS_BASE and PIX_BASE ~~registers~~registers.

For the Real time commands the story is quite the same but for some small differences. The CP will use the INST_INDEX_PORT_RT and INST_DATA_RT register pair instead of the regular ones and there are no wrap around points for real time so the driver must be careful not to overwrite regular shader data. The shared code (shared subroutines) uses the same path as real time.

R400 CP's Views of Instruction Memory

Updated: 11/14/2001
John A. Carey



CP writes code start addresses to appropriate Sub-Blocks so Sequencer knows where to start executing the code.

CP writes code start addresses to appropriate Sub-Blocks so Sequencer knows where to start executing the code.

4. Sequencer Instructions

All control flow instructions and move instructions are handled by the sequencer only. The ALUs will perform NOPs during this time (MOV PV,PV, PS,PS).

5. Constant Stores

5.1 Memory organizations

A likely size for the ALU constant store is 1024x128 bits. The read BW from the ALU constant store is 128 bits/clock and the write bandwidth is 32 bits/clock (directed by the CP bus size not by memory ports).

The maximum logical size of the constant store for a given shader is 256 constants. Or 512 for the pixel/vertex shader pair. The size of the ~~remaping~~re-mapping table is 128 lines (each line addresses 4 constants). The write granularity is 4 constants or 512 bits. It takes 16 clocks to write the four constants.

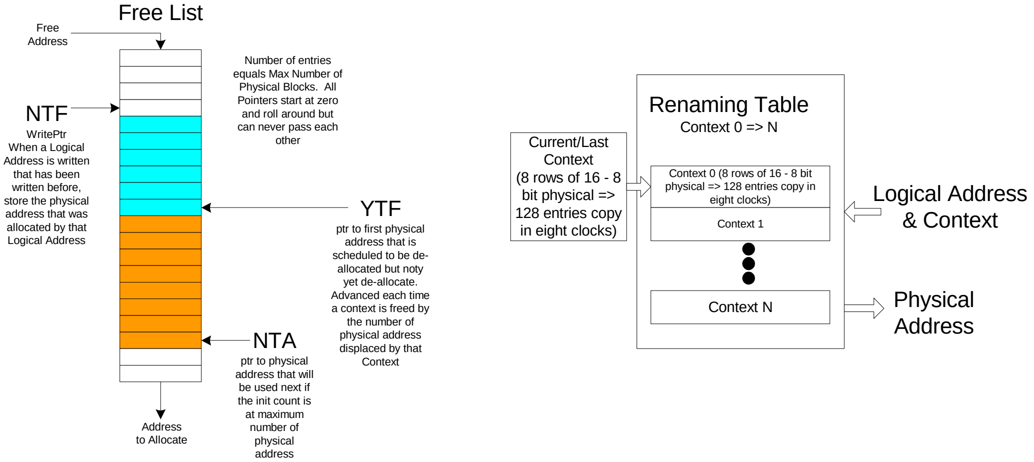
The texture state is also kept in a similar memory. The size of this memory is 192x128. The memory thus holds 128 texture states (192 bits per state). The logical size ~~exposed~~ exposes 32 different states total, which are going to be shared between the pixel and the vertex shader. The size of the ~~remaping~~re-mapping table to for the texture state memory is 16 lines (each line addresses 2 texture state lines in the real memory). The write granularity is 2 texture state lines (or 384 bits). The driver sends 512 bits but the CP ignores the top 128 bits. It thus takes 12 clocks to write the two texture states.

The control flow constant memory doesn't sit behind a renaming table. It is register mapped and thus the driver must reload its content each time there is a state change. Its size is ~~256~~320*32 because it must hold 8 copies of the 32 dwords of control flow constants and the loop construct constants must be aligned.

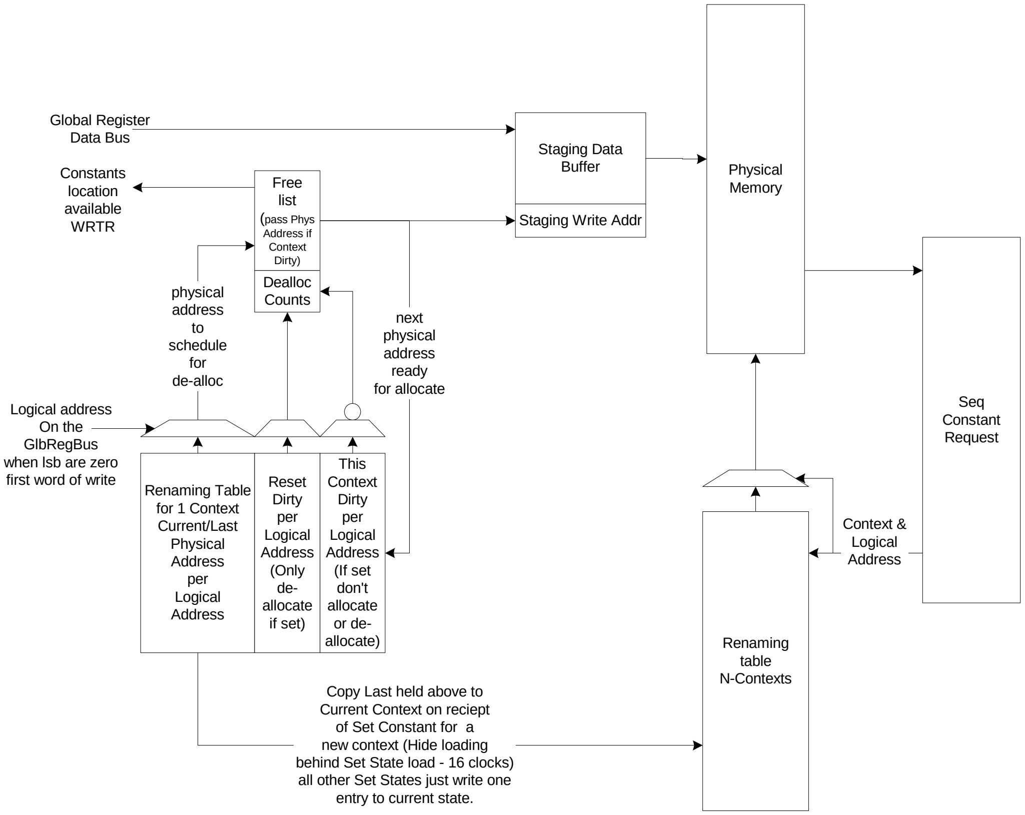
The CP is loading the constant store using the CONST_DATA and CONST_ADDR ~~registers~~registers. It does so by writing to the CONST_ADDR register the logical address for the constant block it wants to update and then writes 16 times to the CONST_DATA register. The CONST_ADDR is auto-incremented on both reads and writes to the CONST_DATA register.

5.2 Management of the ~~remaping~~re-mapping tables

The sequencer is responsible to manage two ~~remaping~~re-mapping tables (one for the constant store and one for the texture state). On a state change (by the driver), the sequencer will broadside copy the contents of its ~~remaping~~re-mapping tables to a new one. We have 8 different ~~remaping~~re-mapping tables we can use concurrently. ~~More details and a diagram to come….~~



Free List
Free Address
Number of entries equals Max Number of Physical Blocks. All Pointers start at zero and roll around but can never pass each other
NTF
WritePtr When a Logical Address is written that has been written before, store the physical address that was allocated by that Logical Address
YTF
ptr to first physical address that is scheduled to be de-allocated but noty yet de-allocate. Advanced each time a context is freed by the number of physical address displaced by that Context
NTA
ptr to physical address that will be used next if the init count is at maximum number of physical address
Address to Allocate
Current/Last Context (8 rows of 16 - 8 bit physical => 128 entries copy in eight clocks)
Renaming Table
Context 0 => N
Context 0 (8 rows of 16 - 8 bit physical => 128 entries copy in eight clocks)
Context 1
Context N
Logical Address & Context
Physical Address



Global Register Data Bus
Staging Data Buffer
Staging Write Addr
Physical Memory
Constants location available WRTR
Free list (pass Phys Address if Context Dirty)
Dealloc Counts
physical address to schedule for de-alloc
next physical address ready for allocate
Logical address On the GlbRegBus when lsb are zero first word of write
Renaming Table for 1 Context Current/Last Physical Address per Logical Address
Reset Dirty per Logical Address (Only de-allocate if set)
This Context Dirty per Logical Address (If set don't allocate or de-allocate)
Seq Constant Request
Context & Logical Address
Renaming table N-Contexts
Copy Last held above to Current Context on reciept of Set Constant for a new context (Hide loading behind Set State load - 16 clocks) all other Set States just write one entry to current state.

5.2.1 *Dirty bits*

Formatted: Bullets and Numbering

Two sets of dirty bits will be maintained per logical address. The first one will be set to zero on reset and set when the logical address is addressed. The second one will be set to zero when ever a new context is written and set for each address written while in this context. The reset dirty is not set, then writing to that logical address will not require de-allocation of whatever address stored in the renaming table. If it is set and the context dirty is not set, then the physical address store needs to be de-allocated and a new physical address is necessary to store the incoming data. If they are both set, then the data will be written into the physical address held in the renaming for the current logical address. No de-allocation or allocation takes place. This will happen when the driver does a set constant twice to the same logical address between context changes. NOTE: It is important to detect and prevent this, failure to do it will allow multiple writes to allocate all physical memory and thus hang because a context will not fit for rendering to start and thus free up space.

5.2.2 *Free List Block*

Formatted: Bullets and Numbering

A free list block that would consist of a counter (called the IFC or Initial Free Counter) that would reset to zero and incremented every time a chunk of physical memory is used until they have all been used once. This counter would be checked each time a physical block is needed, and if the original ones have not been used up, us a new one, else check the free list for an available physical block address. The count is the physical address for when getting a chunk from the counter.

Storage of a free list big enough to store all physical block addresses.

Maintain three pointers for the free list that are reset to zero. The first one we will call NTF (Next To Free). This pointer will identify the next location to write the physical address of a block to be de-allocated. Note: we can never free more physical memory locations than we have. Once recording address the pointer will be incremented to walk the free list like a ring.

The second pointer will be called YTF (Yet To Free). The YTF pointer will be advanced by the number of address chunks de-allocates when a context finishes. The address between the YTF and NTF cannot be reused because they are still in use. But as soon as the context using then is dismissed the YTF will be advanced.

The third pointer will be called NTA (Next To Allocate). This pointer will point will point to the next address that can be used for allocation as long as the NTA does not equal the YTF and the IFC is at its maximum count.

5.2.3 *De-allocate Block*

Formatted: Bullets and Numbering

This block will maintain a free physical address block count for each context. While in current context, a count shall be maintained specifying how many blocks were written into the free list at the NTF pointer. This count will be reset upon reset or when this context is active on the back and different than the previous context. It is actually a count of blocks in the previous context that will no longer be used. This count will be used to advance the NTF pointer to make available the set of physical blocks freed when the previous context was done. This allows the discard or de-allocation of any number of blocks in one clock.

5.2.4 *Operation of Incremental model*

Formatted: Bullets and Numbering

The basic operation of the model would start with the NTF, YTF, NTA pointers in the free list set to zero and the free list counter is set to zero. Also all the dirty bits and the previous context will be initialized to zero. When the first set constants happen, the reset dirty bit will not be set, so we will allocate a physical location from the free list counter because its not at the max value. The data will be written into physical address zero. Both the additional copy of the renaming table and the context zeros of the big renaming table will be updated for the logical address that was written by set start with physical address of 0. This process will be repeated for any logical address that are not dirty until the context changes. If a logical address is hit that has its dirty bits set while in the same context, both dirty bits would be set, so the new data will be over-written to the last physical address assigned for this logical address. When a set constant comes with a different than last context, the previous context stored in the additional renaming table will be copied to the larger renaming table in the current (new) context location. Then the set constant logical address with be loaded with a new physical address during the copy and if the reset dirty was set, the physical address it replaced in the renaming table would be entered at the NTF pointer location on the free list and the NTF will be incremented. The de-allocation counter for the previous context (zero) will be incremented. This as set states come in for this context one of the following will happen:

1.) No dirty bits are set for the logical address being updated. A line will be allocated of the free-list counter or the free list at NTA pointer if NTA != to YTF .

Formatted: Bullets and Numbering

2.) Reset dirty set and Context dirty not set. A new physical address is allocated, the physical address in the renaming table is put on the free list at NTF and it is incremented along with the de-allocate counter for the last context.
3.) Context dirty is set then the data will be written into the physical address specified by the logical address.

This process will continue as long as set states arrive. This block will provide back pressure to the CP when ever he has not free list entries available (counter at max and YTF == NTA). The command stream will keep a count of contexts of constants in use and prevent more than max constants contexts from being sent.
Now preferable when the constant context leaves the last ALU clause it will be sent to this block and compared with the previous context that left. (Init to zero) If they differ than the older context will no longer be referenced and thus can be de-allocated in the physical memory. This is accomplished by adding the number of blocks freed this context to the YTF pointer. This will make all the physical addresses used by this context available to the NTA allocate pointer for future allocation.

This device allows representation of multiple contexts of constants data with N copies of the logical address space. It also allows the second context to be represented as the first set plus some new additional data by just storing the delta's. It allows memory to be efficiently used and when the constants updates are small it can store multiple context. However, if the updates are large, less contexts will be stored and potentially performance will be degraded. Although it will still perform as well as a ring could in this case.

5.3 Constant Store Indexing

In order to do constant store indexing, the sequencer must be loaded first with the indexes (that come from the GPRs). There are 144 wires from the exit of the SP to the sequencer (9 bits pointers x 16 vertexes/clock). Since the data must pass thru the Shader pipe for the float to fixed conversion, there is a latency of 4 clocks (1 instruction) between the time the sequencer is loaded and the time one can index into the constant store. The assembly will look like this

```
MOVA  R1.X,R2.X        // Loads the sequencer with the content of R2.X, also copies the content of R2.X into R1.X
NOP                    // latency of the float to fixed conversion
ADD   R3,R4,C0[R2.X]// Uses the state from the sequencer to add R4 to C0[R2.X] into R3
```

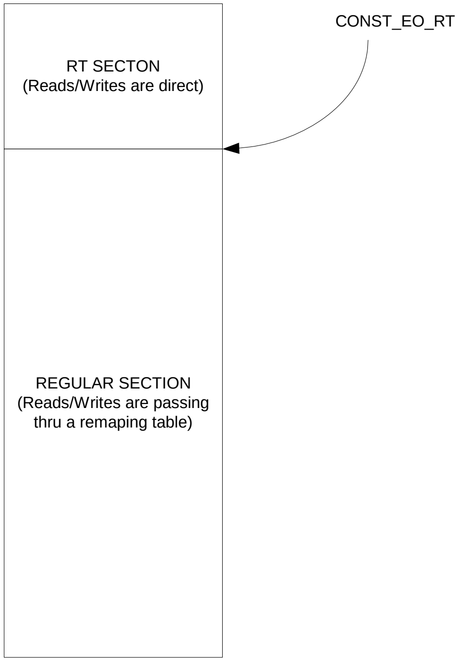
Note that we don't really care about what is in the brackets because we use the state from the MOVA instruction. R2.X is just written again for the sake of simplicity and coherency.

~~5.4~~The storage needed in the sequencer in order to support this feature is 2*64*9 bits = 1152 bits.

Formatted: Bullets and Numbering

~~5.5~~5.4 Real Time Commands

The real time commands constants are written by the CP using the CONST_DATA_RT and CONST_ADDR_RT ~~registers~~GPRs. It works is the same way than when dealing with regular constant loads BUT in this case the CP is not sending a logical address but rather a physical address and the reads are not passing thru the ~~remaping~~re-mapping table but are directly read from the memory. The boundary between the two zones is defined by the CONST_EO_RT control register.



6. Looping and Branches

Loops and branches are planned to be supported and will have to be dealt with at the sequencer level. We plan on supporting constant loops and branches using a control program.

6.1 The controlling state.

The R400 controling state consists of:

Boolean[256:0]
Loop_count[7:0][31:0]
Loop_Start[7:0][31:0]
Loop_Step[7:0][31:0]

That is 256 Booleans and 32 loops.

We have a stack of 4 elements for nested calls of subroutines and 4 loop counters to allow for nested loops.

This state is available on a per shader program basis.

6.2 The Control Flow Program

Examples of control flow programs are located in the R400 programming guide document.

The basic model is as follows:

The render state defined the clause boundaries:
Vertex_shader_fetch[7:0][7:0]   // eight 8 bit pointers to the location where each clauses control program is located
Vertex_shader_alu[7:0][7:0]   // eight 8 bit pointers to the location where each clauses control program is located

Pixel_shader_fetch[7:0][7:0]   // eight 8 bit pointers to the location where each clauses control program is located
Pixel_shader_alu[7:0][7:0]   // eight 8 bit pointers to the location where each clauses control program is located

A pointer value of FF means that the clause doesn't contain any instructions.

The control program for a given clause is executed to completion before moving to another clause, (with the exception of the pick two nature of the alu execution). The control program is the only program aware of the clause boundaries.

The control program has eleven basic instructions:

Execute
Conditional_execute
Conditional_Execute_Predicates
Conditional_jump
Call
Return
Loop_start
Loop_end
End_of_clause
Conditional_End_of_clause
NOP

Execute, causes the specified number of instructions in instruction store to be executed.
Conditional_execute checks a condition first, and if true, causes the specified number of instructions in instruction store to be executed.
Loop_start resets the corresponding loop counter to the start value on the first pass after it checks for the end condition and if met jumps over to a specified address.
Loop_end increments (decrements?) the loop counter and jumps back the specified number of instructions.
Call jumps to an address and pushes the IP counter on the stack. On the return instruction, the IP is popped from the stack.
Conditional_execute_or_Jump executes a block of instructions or jumps to an address is the condition is not met.
Conditional_execute_Predicates executes a block of instructions if all bits in the predicate vectors meet the condition.
End_of_clause marks the end of a clause.
Conditional_End_of_clause marks the end of a clause if the condition is met.
Conditional_jumps jumps to an address if the condition is met.
NOP is a regular NOP

NOTE THAT ALL JUMPS MUST JUMP TO EVEN CFP ADDRESSES since there are two control flow instructions per memory line. Thus the compiler must insert NOPs where needed to align the jumps on even CFP addresses.

Also if the jump is logically bigger than pshader_cntl_size (or vshader_cntl_size) we break the program (clause) and set the debug ~~registers~~registers. If an execute or conditional_execute is lower than cntl_size or bigger than size we also break the program (clause) and set the debug ~~registers~~registers.

We have to fit instructions into 48 bits in order to be able to put two control flow instruction per line in the instruction store.

Note that whenever a field is marked as RESERVED, it is assumed that all the bits of the field are cleared (0).

Execute				
47	46… 42	41 … 24	23 … 12	11 … 0
Addressing	00001	RESERVED	Instruction count	Exec Address

Execute up to 4k instructions at the specified address in the instruction memory.

NOP		
47	46 … 42	41 … 0

Addressing	00010	RESERVED

This is a regular NOP.

Conditio~~n~~nal_Execute							
47	46 … 42	41	40 … 33	32	31 … 24	23 … 12	11 … 0
Addressing	00011	RESERVED	Boolean address	Condition	RESERVED	Instruction count	Exec Address

If the specified ~~boolean~~Boolean (8 bits can address 256 ~~booleans~~Booleans) meets the specified condition then execute the specified instructions (up to 4k instructions)

Conditio~~n~~nal_Execute_Predicates							
47	46 … 42	41 … 35	34 … 33	32	31 … 24	23 … 12	11 … 0
Addressing	00100	RESERVED	Predicate vector	Condition	RESERVED	Instruction count	Exec Address

Check the AND/OR of all current predicate bits. If AND/OR matches the condition execute the specified number of instructions. We need to AND/OR this with the kill mask in order not to consider the pixels that aren't valid.

Loop_Start				
47	46 … 42	41 … 17	16 … 12	11 … 0
Addressing	00101	RESERVED	loop ID	Jump address

Loop Start. Compares the loop ~~count~~iterator with the end value. If loop condition not met jump to the address. Forward jump only. Also computes the index ~~value.~~value. The loop id must match between the start to end, and also indicates which control flow constants should be used with the loop.

Loop_End				
47	46 … 42	41 … 17	16 … 12	11 … 0
Addressing	00111	RESERVED	loop ID	start address

Loop end. Increments the counter by one, compares the loop count with the end value. If loop condition met, continue, else, ~~and~~ jump~~s~~ BACK ~~only~~ to the start of the loop.

The way this is described does not prevent nested loops, and the inclusion of the loop id make this easy to do.

Call			
47	46 … 42	41…12	11 … 0
Addressing	01000	RESERVED	Jump address

Jumps to the specified address and pushes the ~~IP~~ control flow program counter on the stack.

Return		
47	46 … 42	41 … 0
Addressing	01001	RESERVED

Pops the topmost address from the stack and jumps to that address. If nothing is on the stack, the program will just continue to the next instruction.

Conditionnal_Jump							
47	46 … 42	41	40 … 33	32	31	30 … 12	11 … 0
Addressing	01010	RESERVED	Boolean address	Condition	FW only	RESERVED	Jump address

If condition met, jumps to the address. FORWARD jump only allowed if bit 31 set. Bit 31 is only an optimization for the compiler and should NOT be exposed to the API.

Conditional_End_of_Clause					
47	46 … 42	41	40 … 33	32	31 … 0
Addressing	01011	RESERVED	Boolean address	Condition	RESERVED

This is an optimization in the case of very short shaders (where the control flow instruction can't be hidden anymore and thus are not free. In this case, if the condition is met, the clause is ended, else we continue the execution of the clause.

End_of_Clause		
47	46 … 42	41 … 0
Addressing	01011	RESERVED

Marks the end of a clause.

To prevent infinite loops, we will keep 9 bits loop counters instead of 8 (we are only able to loop 256 times). If the counter goes higher than 255 then the loop_end or the loop_start instruction is going to break the loop and set the debug ~~registers~~GPRs.

6.3 Data dependant predicate instructions

Data dependant conditionals will be supported in the R400. The only way we plan to support those is by supporting three vector/scalar predicate operations of the form:

PRED_SETE_# - similar to SETE except that the result is 'exported' to the sequencer.
PRED_SETNE_# - similar to SETNE except that the result is 'exported' to the sequencer.
PRED_SETGT_# - similar to SETGT except that the result is 'exported' to the sequencer
PRED_SETGTE_# - similar to SETGTE except that the result is 'exported' to the sequencer

For the scalar operations only we will also support the two following instructions:
PRED_SETE0_# – SETE0
PRED_SETE1_# – SETE1

The export is a single bit - 1 or 0 that is sent using the same data path as the MOVA instruction. The sequencer will maintain 4 sets of 64 bit predicate vectors (in fact 8 sets because we interleave two programs but only 4 will be exposed) and use it to control the write masking. This predicate is not maintained across clause boundaries. The # sign is used to specify which predicate set you want to use 0 thru 3.

Then we have two conditional execute bits. The first bit is a conditional execute "on" bit and the second bit tells us if we execute on 1 or 0. For example, the instruction:

P0_ADD_# R0,R1,R2

Is only going to write the result of the ADD into those GPRs whose predicate bit is 0. Alternatively, P1_ADD_# would only write the results to the GPRs whose predicate bit is set. The use of the P0 or P1 without precharging the sequencer with a PRED instruction is undefined.

{Issue: do we have to have a NOP between PRED and the first instruction that uses a predicate?}

6.4 HW Detection of PV,PS

Because of the control program, the compiler cannot detect statically dependant instructions. In the case of non-masked writes and subsequent reads the sequencer will insert uses of PV,PS as needed. This will be done by comparing the read address and the write address of consecutive instructions. For masked writes, the sequencer will insert NOPs wherever there is a dependant read/write.

The sequencer will also have to insert NOPs between PRED_SET and MOVA instructions and their uses.

6.5 Register file indexing

Because we can have loops in fetch clause, we need to be able to index into the register file in order to retrieve the data created in a fetch clause loop and use it into an ALU clause. The instruction will include the base address for register indexing and the instruction will contain these controls:

Bit7	Bit 6	
0	0	'absolute register'
0	1	'relative register'
1	0	'previous vector'
1	1	'previous scalar'

In the case of an absolute register we just take the address as is. In the case of a relative register read we take the base address and we add to it the loop_index and this becomes our new address that we give to the shader pipe.

The sequencer is going to keep a loop index computed as such:

Index = Loop_~~counter~~iterator*Loop_~~iterator~~step + Loop_~~init~~start.

The index is going to return 0 if it is out of the range. We loop until loop_iterator = loop_count. Loop_step is a signed value [-128…127].

6.6 Predicated Instruction support for Texture clauses

For texture clauses, we support the following optimization: we keep 1 bit (thus 4 bits for the four predicate vectors) per predicate vector in the reservation stations. A value of 1 means that one ore more elements in the vector have a value of one (thus we have to do the texture fetches for the whole vector. A value of 0 means that no elements in the vector have his predicate bit set and we can thus skip over the texture fetch. **We have to make sure the invalid pixels aren't considered with this optimization.**

6.7 Debugging the Shaders

In order to be able to debug the pixel/vertex shaders efficiently, we provide 2 methods.

6.7.1 *Method 1: Debugging ~~registers~~registers*

Current plans are to expose 2 debugging, or error notification, ~~registers~~registers:
1. address register where the first error occurred
2. count of the number of errors

The sequencer will detect the following groups of errors:
- count overflow
- jump error
  relative jump address > size of the control flow program
  relative jump address > length of the shader program
- constant overflow
- register overflow
- call stack
  call with stack full
  return with stack empty

With two of the errors, a jump error or a register overflow will cause the program to break. In this case, a break means that a clause will halt execution, but allowing further clauses to be executed.

With the other errors, program can continue to run, potentially to worst-case limits.

If indexing outside of the constant range, causing an overflow error, the hardware is specified to return the value with an index of 0. This could be exploited to generate error tokens, by reserving and initializing the 0th register (or constant) for errors.

{ISSUE : Interrupt to the driver or not?}

6.7.2 *Method 2: Exporting the values in the GPRs (12)*

The sequencer will have a count register and an address register for this mode and 3 bits per clause specifying the execution mode for each clause. The modes can be :

1) Normal
2) Debug Kill
3) Debug Addr + Count

Under the normal mode execution follows the normal course. Under the kill mode, all control flow instructions are executed but all normal shader instructions of the clause are replaced by NOPs. Only debug_export instructions of clause 7 will be executed under the debug kill setting. Under the other mode, normal execution is done until we reach an address specified by the address register and instruction count (useful for loops) specified by the count register. After we have hit the instruction n times (n=count) we switch the clause to the kill mode.

Under the debug mode (debug kill OR debug Addr + count), it is assumed that clause 7 is always exporting 12 debug vectors and that all other exports to the SX block (position, color, z, ect) will been turned off (changed into NOPs) by the sequencer (even if they occur before the address stated by the ADDR debug register).

7. Pixel Kill Mask

A vector of 64 bits is kept by the sequencer per group of pixels/vertices. Its purpose is to optimize the texture fetch requests and allow the shader pipe to kill pixels using the following instructions:

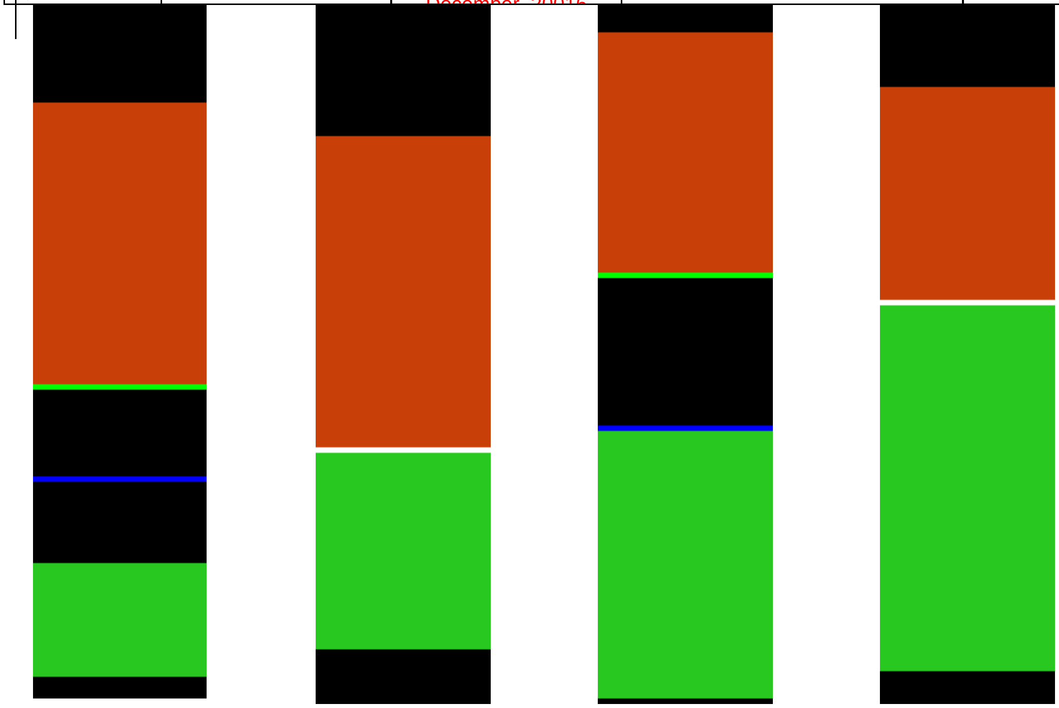
MASK_SETE
MASK_SETNE
MASK_SETGT
MASK_SETGTE

8. Multipass vertex shaders (HOS)

Multipass vertex shaders are able to export from the 6 last clauses but to memory ONLY.

9. Register file allocation

The register file allocation for vertices and pixels can either be static or dynamic. In both cases, the register file in managed using two round robins (one for pixels and one for vertices). In the dynamic case the boundary between pixels and vertices is allowed to move, in the static case it is fixed to VERTEX_REG_SIZE for vertices and 256-VERTEX_REG_SIZE for pixels.



Above is an example of how the algorithm works. Vertices come in from top to bottom; pixels come in from bottom to top. Vertices are in orange and pixels in green. The blue line is the tail of the vertices and the green line is the tail of the pixels. Thus anything between the two lines is shared. When pixels meets vertices the line turns white and the boundary is static until both vertices and pixels share the same “unallocated bubble”. Then the boundary is allowed to move again.

10. Fetch Arbitration

The fetch arbitration logic chooses one of the 8 potentially pending fetch clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute. Once chosen, the clause state machine will send one 2x2 fetch per clock (or 4 fetches in one clock every 4 clocks) until all the fetch instructions of the clause are sent. This means that there cannot be any dependencies between two fetches of the same clause.

The arbitrator will not wait for the fetches to return prior to selecting another clause for execution. The fetch pipe will be able to handle up to X(?) in flight fetches and thus there can be a fair number of active clauses waiting for their fetch return data.

11. ALU Arbitration

ALU arbitration proceeds in almost the same way than fetch arbitration. The ALU arbitration logic chooses one of the 8 potentially pending ALU clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute. There are two ALU ~~arbitrers~~arbiters, one for the even clocks and one for the odd clocks. For exa~~e~~mple, here is the sequencing of two interleaved ALU clauses (E and O stands for Even and Odd sets of 4 clocks):

Einst0 Oinst0 Einst1 Oinst1 Einst2 Oinst2 Einst0 Oinst3 Einst1 Oinst4 Einst2 Oinst0…
Proceeding this way hides the latency of 8 clocks of the ALUs. Also note that the interleaving also occurs across clause boundaries.

12. Handling Stalls

When the output file is full, the sequencer prevents the ALU arbitration logic ~~to select~~from selecting the last clause (this way nothing can exit the shader pipe until there is place in the output file. If the packet is a vertex packet and the position buffer is full (POS_FULL) then the sequencer also prevents a thread ~~to enter~~from entering the exporting clause (3?). The sequencer will set the OUT_FILE_FULL signal n clocks before the output file is actually full and thus the ALU arbiter will be able read this signal and act accordingly by not preventing exporting clauses to proceed.

13. Content of the reservation station FIFOs

The reservation FIFOs contain the state of the vector of pixels and vertices. We have two sets of those: one for pixels, and one for vertices. They contain 3 bits of Render State 7 bits for the base address of the GPRs, some bits for LOD correction and coverage mask information in order to fetch fetch for only valid pixels, the quad address~~ and 1 bit to specify if the vector is of pixels or vertices~~. ~~Since pixels and vertices are kept in order in the shader pipe, we only need two fifos (one for vertices and one for pixels) deep enough to cover the shader pipe latency. This size will be determined later when we will know the size of the small fifos between the reservation stations.~~

14. The Output File

The output file is where pixels are put before they go to the RBs. The write BW to this store is 256 bits/clock. Just before this output file are staging ~~registers~~registers with write BW 512 bits/clock and read BW 256 bits/clock. The staging ~~registers~~registers are 4x128 (and there are 16 of those on the whole chip).

15. IJ Format

The IJ information sent by the PA is of this format on a per quad basis:

We have a vector of IJ's (one IJ per pixel at the centroid of the fragment or at the center of the pixel depending on the mode bit). The interpolation is done at a different precision across the 2x2. The upper left pixel's parameters are always interpolated at full 20x24 mantissa precision. Then the result of the interpolation along with the difference in IJ in reduced precision is used to interpolate the parameter for the other three pixels of the 2x2. Here is how we do it:

Assuming P0 is the interpolated parameter at Pixel 0 having the barycentric coordinates I(0), J(0) and so on for P1,P2 and P3. Also assuming that A is the parameter value at V0 (interpolated with I), B is the parameter value at V1 (interpolated with J) and C is the parameter value at V2 (interpolated with (1-I-J).

$$\Delta 01I = I(1) - I(0)$$
$$\Delta 01J = J(1) - J(0)$$
$$\Delta 02I = I(2) - I(0)$$
$$\Delta 02J = J(2) - J(0)$$
$$\Delta 03I = I(3) - I(0)$$
$$\Delta 03J = J(3) - J(0)$$

P0	P1
P2	P3

$$P0 = C + I(0)*(A - C) + J(0)*(B - C)$$
$$P1 = P0 + \Delta 01I*(A - C) + \Delta 01J*(B - C)$$
$$P2 = P0 + \Delta 02I*(A - C) + \Delta 02J*(B - C)$$
$$P3 = P0 + \Delta 03I*(A - C) + \Delta 03J*(B - C)$$

P0 is computed at 20x24 mantissa precision and P1 to P3 are computed at 8X24 mantissa precision. So far no visual degradation of the image was seen using this scheme.

Multiplies (Full Precision): 2
Multiplies (Reduced precision): 6
Subtracts 19x24 (Parameters): 2
Adds: 8

FORMAT OF P0's IJ : Mantissa 20 Exp 4 for I + Sign
Mantissa 20 Exp 4 for J + Sign

FORMAT of Deltas (x3): Mantissa 8 Exp 4 for I + Sign
Mantissa 8 Exp 4 for J + Sign

Total number of bits : 20*2 + 8*6 + 4*8 + 4*2 = 128

~~The Deltas have a leading 1, the Full precision IJs don't. This means that in the case of the deltas we MUST be able to shift 8 right (exponent value of 0 means number = 0, exponent value of 1 means shift right 8).~~All numbers are kept using the un-normalized floating point convention: if exponent is different than 0 the number is normalized if not, then the number is un-normalized. ~~This means that t~~The maximum range for the IJs (Full precision) is +/- 63 and the range for the Deltas is +/- 127.

15.1 Interpolation of constant attributes

Because of the floating point imprecision, we need to take special provisions if all the interpolated terms are the same or if two of the barycentric coordinates are the same.

We start with the premise that if A = B and B = C and C = A, then P0,1,2,3 = A. Since one or more of the IJ terms may be zero, so we extend this to:

```
if (A=B and B=C and C=A)
   P0,1,2,3 = A;
else if ((I = 0) or (J = 0)) and
         ((J = 0) or (1-I-J = 0)) and
         ((1-J-I = 0) or (I = 0))) {
              if(I != 0) {
                 P0 = A;
              } else if(J != 0) {
                 P0 = B;
              } else {
                 P0 = C;
             }
           //rest of the quad interpolated normally
}
else
{
           normal interpolation
}
```

16. The parameter cache

The parameter cache is where the vertex shaders export their data. It consists of 16 128x128 memories (1R/1W). The reuse engine will make it so that all vertexes of a given primitive will hit different memories.

17. Vertex position exporting

On clause 3 the vertex shader can export to the PA both the vertex position and the point sprite. It can also do so at clause 7 if not done at clause 3. The storage needed to perform the position export is at least 64x128 memories for the position and 64x32 memories for the sprite size. It is going to be taken in the pixel output fifo from the SX blocks. The clause where the position export occurs is specified by the EXPORT_LATE register. If turned on, it means that the export is going to occur at ALU clause 7 if unset position export occurs at clause 3.

18. Exporting Arbitration

Here are the rules for co-issuing exporting ALU clauses.

1) Position exports and position exports cannot be co-issued.
2) Position exports and memory exports cannot be co-issued.
3) Position exports and Z/Color exports cannot be co-issued.
4) Memory exports and Z/Color exports cannot be co-issued.
5) Memory exports and memory exports cannot be co-issued.
6) Z/color exports and Z/color exports cannot be co-issued.
7) Parameter exports and Z/Color exports CAN be co-issued.
8) Parameter exports and parameter exports CAN be co-issued.
9) Parameter exports and memory exports CAN be co-issued.

19. Export Types

The export type (or the location where the data should be put) is specified using the destination address field in the ALU instruction. Here is a list of all possible export modes:

19.1 Vertex Shading

0:15	- 16 parameter cache
16:31	- Empty (Reserved?)
32:43	- 12 vertex exports to the frame buffer and index
44:47	- Empty
48:59	- 12 debug export (interpret as normal vertex export)
60	- export addressing mode
61	- Empty
62	- sprite size export that goes with position export (point_h,point_w,edgeflag,misc)
63	- position

19.2 Pixel Shading

0	- Color for buffer 0 (primary)
1	- Color for buffer 1
2	- Color for buffer 2
3	- Color for buffer 3
4:7	- Empty
8	- Buffer 0 Color/Fog (primary)
9	- Buffer 1 Color/Fog
10	- Buffer 2 Color/Fog
11	- Buffer 3 Color/Fog
12:15	- Empty
16:31	- Empty (Reserved?)
32:43	- 12 exports for multipass pixel shaders.
44:47	- Empty
48:59	- 12 debug exports (interpret as normal pixel export)
60	- export addressing mode
61:62	- Empty
63	- Z for primary buffer (Z exported to 'alpha' component)

20. Special Interpolation modes

20.1 Real time commands

We are unable to use the parameter memory since there is no way for a command stream to write into it. Instead we need to add three 16x128 memories (one for each of three vertices x 16 interpolants). These will be mapped onto the register bus and written by type 0 packets, and output to the the parameter busses (the sequencer and/or PA need to be able to address the realtime parameter memory as well as the regular parameter store. For higher performance we should be able able to view them as two banks of 16 and do double buffering allowing one to be loaded, while the other is rasterized with. Most overlay shaders will need 2 or 4 scalar coordinates, one option might be to restrict the memory to 16x64 or 32x64 allowing only two interpolated scalars per cycle, the only problem I see with this is, if we view support for 16 vector-4 interpolants important (true only if we map Microsoft's high priority stream to the realtime stream), then the PA/sequencer need to support a realtime-specific mode where we need to address 32 vectors of parameters instead of 16. This mode is triggered by the primitive type: REAL TIME.

20.2 Sprites/ XY screen coordinates/ FB information

When working with sprites, one may want to overwrite the parameter 0 with SC generated data. Also, XY screen coordinates may be needed in the shader program. This functionality is controlled by the gen_I0 register (in SQ) in conjunction with the SND_XY register (in SC). Also it is possible to send the faceness information (for OGL front/back special operations) to the shader using the same control ~~registers~~register. Here is a list of all the modes and how they interact together:

Gen_st is a bit taken from the interface between the SC and the SQ. This is the MSB of the primitive type. If the bit is set, it means we are dealing with Point AA, Line AA or sprite and in this case the vertex values are going to generated between 0 and 1.

Param_Gen_I0 disable, snd_xy disable, no gen_st – I0 = No modification
Param_Gen_I0 disable, snd_xy disable, gen_st – I0 = No modification
Param_Gen_I0 disable, snd_xy enable, no gen_st – I0 = No modification
Param_Gen_I0 disable, snd_xy enable, gen_st – I0 = No modification
Param_Gen_I0 enable, snd_xy disable, no gen_st – I0 = garbage, garbage, garbage, faceness
Param_Gen_I0 enable, snd_xy disable, gen_st – I0 = garbage, garbage, s, t
Param_Gen_I0 enable, snd_xy enable, no gen_st – I0 = screen x, screen y, garbage, faceness
Param_Gen_I0 enable, snd_xy enable, gen_st – I0 = screen x, screen y, s, t

20.3 Auto generated counters

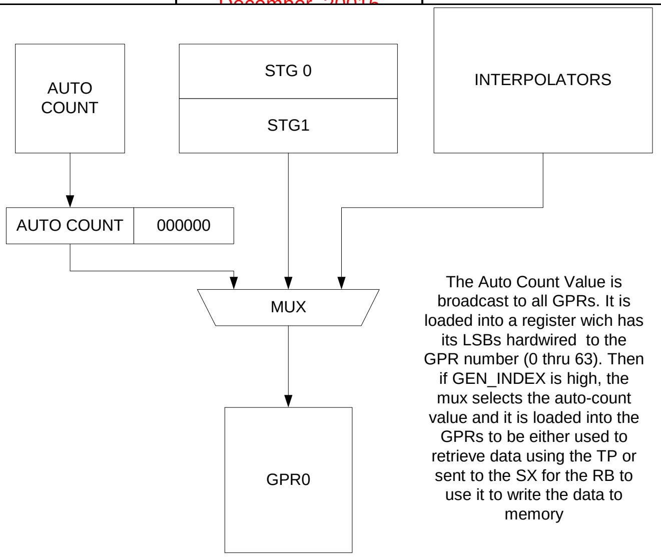
In the cases we are dealing with multipass shaders, the sequencer is going to generate a vector count to be able to both use this count to write the 1st pass data to memory and then use the count to retrieve the data on the 2nd pass. The count is always generated in the same way but it is passed to the shader in a slightly different way depending on the shader type (pixel or vertex). This is toggled on and off using the GEN_INDEX register. While there is only one count broadcast to the ~~registers~~GPRs, the LSB are hardwired to specific values making the index different for all elements in the vector.

20.3.1 *Vertex shaders*

In the case of vertex shaders, if GEN_INDEX is set, the data will be put into the x field of the third register (it means that the compiler must allocate 3 GPRs in all multipass vertex shader modes).

20.3.2 *Pixel shaders*

In the case of pixel shaders, if GEN_INDEX is set, the data will be put in the x field of the 2nd register (I1.x).



The Auto Count Value is broadcast to all GPRs. It is loaded into a register wich has its LSBs hardwired to the GPR number (0 thru 63). Then if GEN_INDEX is high, the mux selects the auto-count value and it is loaded into the GPRs to be either used to retrieve data using the TP or sent to the SX for the RB to use it to write the data to memory

21. State management

Every clock, the sequencer will report to the CP the oldest states still in the pipe. These are the states of the programs as they enter the last ALU clause.

21.1 Parameter cache synchronization

Formatted: Bullets and Numbering

In order for the sequencer not to begin a group of pixels before the associated group of vertices has finished, the sequencer will keep a 6 bit count per state (for a total of 8 counters). These counters are initialized to 0 and every time a vertex shader exports its data TO THE PARAMETER CACHE, the corresponding pointer is incremented. When the SC sends a new vector of pixels with the SC_SQ_new_vector bit asserted, the sequencer will first check if the count is greater than 0 before accepting the transmission (it will in fact accept the transmission but then lower its ready to receive). Then the sequencer waits for the count to go to one and decrements it. The sequencer can then issue the group of pixels to the interpolators. Every time the state changes, the new state counter is initialized to 0.

22. XY Address imports

The SC will be able to send the XY addresses to the GPRs. It does so by interleaving the writes of the IJs (to the IJ buffer) with XY writes (to the XY buffer). Then when writing the data to the GPRs, the sequencer is going to interpolate the IJ data or pass the XY data thru a Fix→float converter and expander and write the converted values to the GPRs. The Xys are currently SCREEN SPACE COORDINATES. The values in the XY buffers will wrap. See section 20.2 for details on how to control the interpolation in this mode.

22.1 Vertex indexes imports

In order to import vertex indexes, we have 64x2x96 staging ~~registers~~registers. These are loaded one at a time by the VGT block. They are loaded in floating point format and can be transferred in 4 or 8 clocks to the GPRs.

23. ~~Registers~~Registers

23.1 Control

REG_DYNAMIC Dynamic allocation (pixel/vertex) of the register file on or off.

REG_SIZE_PIX Size of the register file's pixel portion (minimal size when dynamic allocation turned on)

REG_SIZE_VTX Size of the register file's vertex portion (minimal size when dynamic allocation turned on)

ARBITRATION_POLICY policy of the arbitration between vertexes and pixels

INST_STORE_ALLOC interleaved, separate

INST_BASE_VTX start point for the vertex instruction store (RT always ends at vertex_base and Begins at 0)

INST_BASE_PIX start point for the pixel shader instruction store

ONE_THREAD debug state register. Only allows one program at a time into the GPRs

ONE_ALU debug state register. Only allows one ALU program at a time to be executed (instead of 2)

INSTRUCTION_~~INDEX~~ ~~_PORT~~ADDR This is where the CP puts the base address of the instruction writes and type (auto-incremented on reads/writes)

INSTRUCTION_DATA This is where the CP puts the actual data going to the instruction memory

CONSTANT_DATA This is where the CP puts constant data (32 bits)

CONSTANT_ADDR This is where the CP puts the logical constant address (9 bits)

INSTRUCTION_~~INDEX~~ ~~PORT~~ADDR_RT This is where the CP puts the base address of the instruction writes and type for Real Time (auto-incremented on reads/writes)

INSTRUCTION_DATA_RT This is where the CP puts the actual data going to the instruction memory for Real Time

CONSTANT_DATA_RT This is where the CP puts constant data for Real Time (32 bits)

CONSTANT_ADDR_RT This is where the CP puts the logical constant address for Real Time (9 bits)

CONSTANT_EO_RT This is the size of the space reserved for real time in the constant store (from 0 to CONSTANT_EO_RT). The ~~remaping~~re-mapping table operates on the rest of the memory

EXPORT_LATE Controls whether or not we are exporting position from clause 3. If set, position exports occur at clause 7.

23.2 Context

VS_FETCH_{0…7} eight 8 bit pointers to the location where each clauses control program is located

VS_ALU_{0…7} eight 8 bit pointers to the location where each clauses control program is located

PS_FETCH_{0…7} eight 8 bit pointers to the location where each clauses control program is located

PS_ALU_{0…7} eight 8 bit pointers to the location where each clauses control program is located

PS_BASE base pointer for the pixel shader in the instruction store

VS_BASE base pointer for the vertex shader in the instruction store

VS_CF_SIZE size of the vertex shader (# of instructions in control program/2)

PS_CF_SIZE size of the pixel shader (# of instructions in control program/2)

PS_SIZE size of the pixel shader (cntl+instructions)

VS_SIZE size of the vertex shader (cntl+instructions)

PS_NUM_REG number of ~~registers~~GPRs to allocate for pixel shader programs

VS_NUM_REG number of ~~registers~~GPRs to allocate for vertex shader programs

PARAM_SHADE One 16 bit register specifying which parameters are to be gouraud shaded (0 = flat, 1 = gouraud)

PARAM_WRAP 64 bits: for which parameters (and channels (xyzw)) do we do the cyl wrapping (0=linear, 1=cylindrical).

PS_EXPORT_MODE 0xxxx : Normal mode
1xxxx : Multipass mode
If normal, bbbz where bbb is how many colors (0-4) and z is export z or not
If multipass 1-12 exports for color.

VS_EXPORT_MASK which of the last 6 ALU clauses is exporting (multipass only)

VS_EXPORT_MODE 0: position (1 vector), 1: position (2 vectors), 3:multipass

VS_EXPORT _COUNT_{0…6} Six 4 bit counters representing the # of interpolated parameters exported in clause 7 (located in VS_EXPORT_COUNT_6) OR
# of exported vectors to memory per clause in multipass mode (per clause)

PARAM_GEN_I0	Do we overwrite or not the parameter 0 with XY data and generated T and S values
GEN_INDEX	Auto generates an address from 0 to XX. Puts the results into R1 for pixel shaders and R3 for vertex shaders
CONST_BASE_VTX (9 bits)	Logical Base address for the constants of the Vertex shader
CONST_BASE_PIX (9 bits)	Logical Base address for the constants of the Pixel shader
CONST_SIZE_PIX (8 bits)	Size of the logical constant store for pixel shaders
CONST_SIZE_VTX (8 bits)	Size of the logical constant store for vertex shaders
INST_PRED_OPTIMIZE	Turns on the predicate bit optimization (if of, conditional_execute_predicates is always executed).
CF_BOOLEANS	256 boolean bits
CF_LOOP_COUNT	32x8 bit counters (number of times we traverse the loop)
CF_LOOP_START	32x8 bit counters (init value used in index computation)
CF_LOOP_STEP	32x8 bit counters (step value used in index computation)

24. DEBUG ~~registers~~Registers

24.1 Context

DB_PROB_ADDR	instruction address where the first problem occurred
DB_PROB_COUNT	number of problems encountered during the execution of the program
DB_INST_COUNT	instruction counter for debug method 2
DB_BREAK_ADDR	break address for method number 2
DB_CLAUSE _MODE_ALU_{0…7}	clause mode for debug method 2 (0: normal, 1: addr, 2: kill)
DB_CLAUSE _MODE_FETCH_{0…7}	clause mode for debug method 2 (0: normal, 1: addr, 2: kill)

25. Interfaces

25.1 External Interfaces

Whenever an x is used, it means that the bus is broadcast to all units of the same name. For example, if a bus is named SQ→SPx it means that SQ is going to broadcast the same information to all SP instances.

~~25.1.1 SC to SP : IJ bus~~

Formatted: Bullets and Numbering

~~This is a bus that sends the IJ information to the IJ fifos on the top of each shader pipe. At the same time the control information goes to the sequencer. There are 4 of these buses over the whole chip (SP0 thru 3)~~

~~25.1.2~~25.1.1 *SC to SQ : IJ Control bus*

Formatted: Bullets and Numbering

This is the control information sent to the sequencer in order to control the IJ fifos and all other information needed to execute a shader program on the sent pixels. This information is sent over 2 clocks, if SENDXY is asserted the next control packet is going to be ignored and XY information is going to be sent on the IJ bus (for the quads that where just sent). All pixels from the group of quads are from the same primitive, all quads of a vector are from the same render state.

Name	Direction	Bits	Description
SC_SQ_q_wr_mask	SC→SQ	4	Quad Write mask left to right
SC_SQ_lod_correct	SC→SQ	24	LOD correction per quad (6 bits per quad)
SC_SQ_flat_vertex	SC→SQ	2	Provoking vertex for flat shading
SC_SQ_param_ptr0	SC→SQ	11	P Store pointer for vertex 0
SC_SQ_param_ptr1	SC→SQ	11	P Store pointer for vertex 1
SC_SQ_param_ptr2	SC→SQ	11	P Store pointer for vertex 2
SC_SQ_end_of_vect	SC→SQ	1	End of the vector
SC_SQ_store_dealloc	SC→SQ	1	Deallocation token for the P Store
SC_SQ_state	SC→SQ	3	State/constant pointer ~~(6*3+3)~~
SC_SQ_valid_pixel	SC→SQ	16	Valid bits for all pixels
SC_SQ_null_prim	SC→SQ	1	Null Primitive (for PC deallocation purposes)
SC_SQ_end_of_prim	SC→SQ	1	End Of the primitive
SC_SQ_fbface	SC→SQ	1	Front face = 1, back face = 0
SC_SQ_send_xy	SC→SQ	1	Sending XY information [XY information is going to be sent on the next clock]
SC_SQ_prim_type	SC→SQ	3	Real time command need to load tex cords from alternate buffer. Line AA, Point AA and Sprite reads their parameters from GEN_T and GEN_S ~~registers~~GPRs. 000 : Normal 011 : Real Time 100 : Line AA 101 : Point AA 110 : Sprite
SC_SQ_new_vector	SC→SQ	1	This primitive comes from a new vector of vertices. Make sure that the corresponding vertex shader has finished before starting the group of pixels.
SC_SQ_RTRn	SQ→SC	1	Stalls the PA in n clocks
SC_SQ_RTS	SC→SQ	1	SC ready to send data

~~25.1.3~~25.1.2 *SQ to SP: Interpolator bus*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SPx_interp_prim_type	SQ→SPx	3	Type of the primitive 000 : Normal 011 : Real Time 100 : Line AA 101 : Point AA 110 : Sprite
SQ_SPx_interp_flat_vtx	SQ→SPx	2	Provoking vertex for flat shading
SQ_SPx_interp_flat_gouraud	SQ→SPx	1	Flat or gouraud shading
SQ_SPx_interp_cyl_wrap	SQ→SPx	4	Wich parameter needs to be cylindrical wrapped
SQ_SPx_interp_ijline	SQ→SPx	2	Line in the IJ/XY buffer to use to interpolate
SQ_SPx_interp_buff_swap	SQ→SPx	1	Swap the IJ/XY buffers at the end of the interpolation
SQ_SPx_interp_gen_I0	SQ→SPx	1	Generate I0 or not. This tells the interpolators not to use the parameter cache but rather overwrite the data with interpolated 1 and 0. Overwrite if gen_I0 is high.

~~25.1.4~~25.1.3 *SQ to SP: GPR Input Mux select*

Formatted: Bullets and Numbering

This interface is synchronized with the Interpolator bus. This controls the input mux to the GPRs. The three types of data are: generated index, Interpolated data, vertex index data (coming from the staging ~~registers~~registers).

Name	Direction	Bits	Description
SQ_SPx_data_type	SQ→SPx	2	00: Interpolated data 01: Staging register data 1x: Count
SQ_SPx_index_count	SQ→SPx	12?	Index count, common for all shader pipes
SQ_SPx_stage_addr	SQ→SPx	1	Staging register address 0: First staging register 1: second staging register

~~25.1.5 SQ to SPx: Parameter cache write control~~

Formatted: Bullets and Numbering

~~25.1.6~~25.1.4 SQ to SP: Parameter Cache Read control bus

The four following interfaces (SQ→SP, SQ→SX,SP→SX and SX→Interpolators) are all SYNCHRONIZED together.

Name	Direction	Bits	Description
SQ_SPx_ptr0	SQ→SPx	9	Pointer of PC
SQ_SPx_ptr1	SQ→SPx	9	Pointer of PC
SQ_SPx_ptr2	SQ→SPx	9	Pointer of PC
SQ_SP0_read_ena	SQ→SP0	4	Read enables for the 4 memories in the SP0
SQ_SP1_read_ena	SQ→SP1	4	Read enables for the 4 memories in the SP1
SQ_SP2_read_ena	SQ→SP2	4	Read enables for the 4 memories in the SP2
SQ_SP3_read_ena	SQ→SP3	4	Read enables for the 4 memories in the SP3

~~25.1.7~~25.1.5 SQ to SX: Parameter Cache Mux control Bus

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SXx_mux0	SQ→SXx	4	Mux control for PC (4 MSbs of Pointer)
SQ_SXx_mux1	SQ→SXx	4	Mux control for PC (4 MSbs of Pointer)
SQ_SXx_mux2	SQ→SXx	4	Mux control for PC (4 MSbs of Pointer)

~~25.1.8 SP to SX: Parameter data~~

Formatted: Bullets and Numbering

~~25.1.9 SX to Interpolators: Parameter Cache Return bus~~

Formatted: Bullets and Numbering

~~25.1.10~~25.1.6 SQ to SP~~0~~: Staging Register Data

Name	Direction	Bits	Description
SQ_SP0_vgt_vsisr_data	SQ→SP0	96	Pointers of indexes or HOS surface information
SQ_SP0_vgt_vsisr_double	SQ→SP0	1	0: Normal 96 bits per vert 1: double 192 bits per vert
SQ_SP0_data_valid	SQ→SP0	1	Data is valid
SQ_SP1_vgt_vsisr_data	SQ→SP1	96	Pointers of indexes or HOS surface information
SQ_SP1_vgt_vsisr_double	SQ→SP1	1	0: Normal 96 bits per vert 1: double 192 bits per vert
SQ_SP1_data_valid	SQ→SP1	1	Data is valid
SQ_SP2_vgt_vsisr_data	SQ→SP2	96	Pointers of indexes or HOS surface information
SQ_SP2_vgt_vsisr_double	SQ→SP2	1	0: Normal 96 bits per vert 1: double 192 bits per vert
SQ_SP2_data_valid	SQ→SP2	1	Data is valid
SQ_SP3_vgt_vsisr_data	SQ→SP3	96	Pointers of indexes or HOS surface information
SQ_SP3_vgt_vsisr_double	SQ→SP3	1	0: Normal 96 bits per vert 1: double 192 bits per vert
SQ_SP3_data_valid	SQ→SP3	1	Data is valid

~~25.1.11~~25.1.7 PA to SQ : Vertex interface

Formatted: Bullets and Numbering

~~25.1.11.1~~25.1.7.1 Interface Signal Table

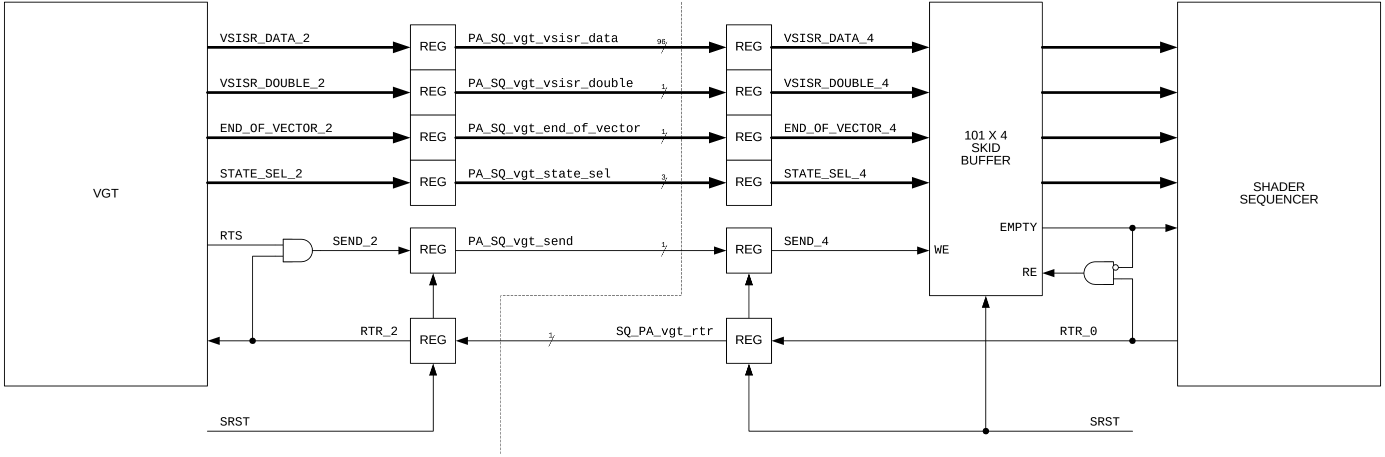
The area difference between the two methods is not sufficient to warrant complicating the interface or the state requirements of the VSISRs. **Therefore, the POR for this interface is that the VGT will transmit the data to the**

VSISRs (via the Shader Sequencer) in full, 32-bit floating-point format. The VGT can transmit up to six 32-bit floating-point values to each VSISR where four or more values require two transmission clocks. The data bus is 96 bits wide.

Name	Bits	Description
PA_SQ_vgt_vsisr_data	96	Pointers of indexes or HOS surface information
PA_SQ_vgt_vsisr_double	1	0: Normal 96 bits per vert 1: double 192 bits per vert
PA_SQ_vgt_end_of_vector	1	Indicates the last VSISR data set for the current process vector (for double vector data, "end_of_vector" is set on the second vector)
PA_SQ_vgt_state	3	Render State (6*3+3 for constants). This signal is guaranteed to be correct when “PA_SQ_vgt_end_of_vector” is high.
PA_SQ_vgt_send	1	Data on the VGT_SQ is valid receive (see write-up for standard R400 SEND/RTR interface handshaking)
SQ_PA_vgt_rtr	1	Ready to receive (see write-up for standard R400 SEND/RTR interface handshaking)

~~25.1.11.2~~25.1.7.2 Interface Diagrams

Formatted: Bullets and Numbering



VGT
VSISR_DATA_2
VSISR_DOUBLE_2
END_OF_VECTOR_2
STATE_SEL_2
RTS
SEND_2
RTR_2
SRST
REG
PA_SQ_vgt_vsisr_data
96
PA_SQ_vgt_vsisr_double
1
PA_SQ_vgt_end_of_vector
1
PA_SQ_vgt_state_sel
3
PA_SQ_vgt_send
1
SQ_PA_vgt_rtr
1
VSISR_DATA_4
VSISR_DOUBLE_4
END_OF_VECTOR_4
STATE_SEL_4
SEND_4
101 X 4
SKID
BUFFER
EMPTY
WE
RE
RTR_0
SRST
SHADER
SEQUENCER

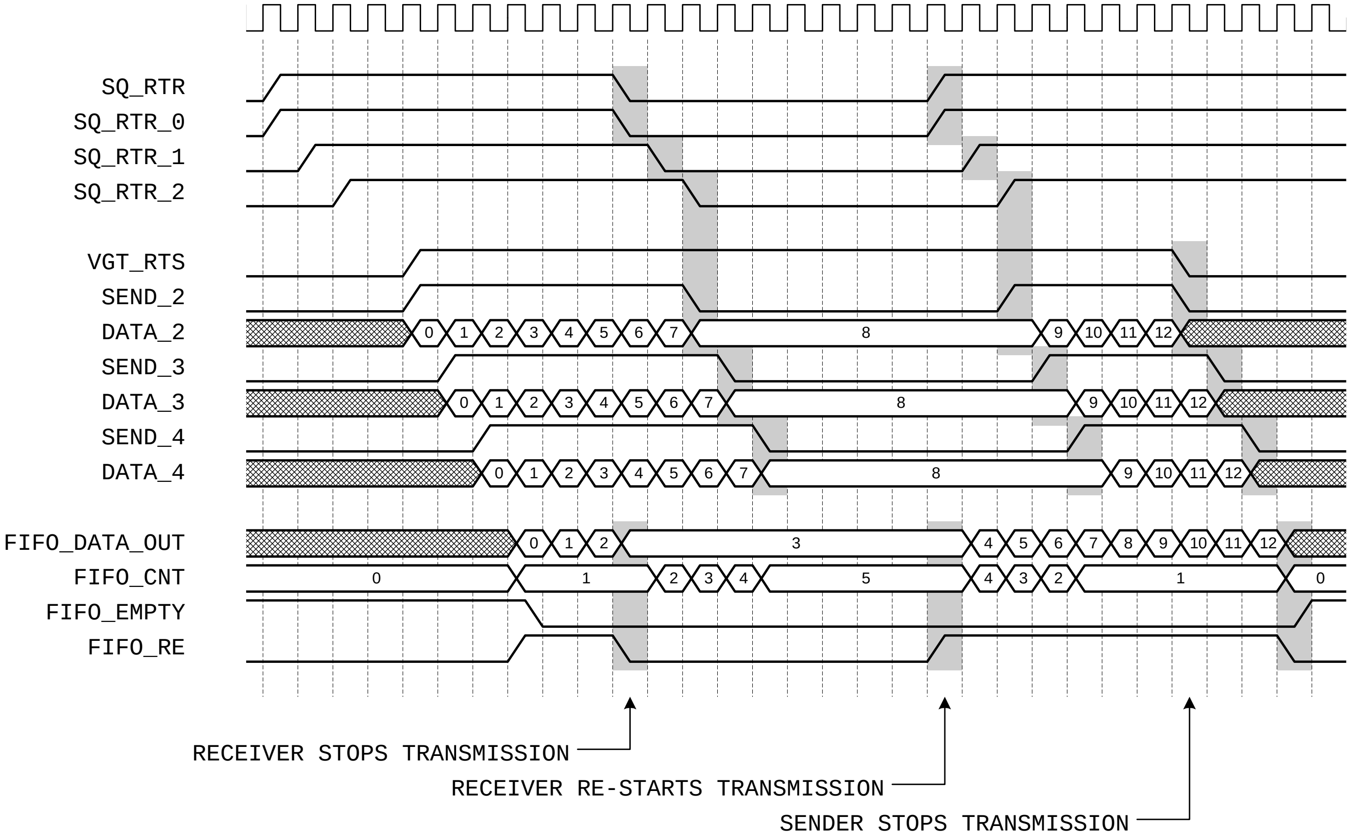


Figure 1. Detailed Logical Diagram for PA_SQ_vgt Interface.

~~25.1.12~~25.1.8 *SQ to CP: State report*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_CP_vrtx_ state	SEQ→CP	3	Oldest vertex state still in the pipe
SQ_CP_pix_state	SEQ→CP	3	Oldest pixel state still in the pipe

~~25.1.13 *SP to SX : Pixel/Vertex write to SX*~~

Formatted: Bullets and Numbering

~~25.1.14~~25.1.9 *SQ to SX: Control bus*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SXx_exp_Pixel	SQ→SXx	1	1: Pixel 0: Vertex
SQ_SXx_exp_start	SQ→SXx	1	Raised to indicate that the SQ is starting an export
SQ_SXx_exp_Clause	SQ→SXx	3	Clause number, which is needed for vertex clauses
SQ_SXx_exp_State	SQ→SXx	3	State ID, which is needed for vertex clauses

These fields are sent synchronously with SP export data, described in SP0→SX0 interface
{ISSUE: Where are the PC pointers}

~~25.1.15~~25.1.10 *SX to SQ : Output file control*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SXx_SQ_Export_count_rdy	SXx→SQ	1	Raised by SX0 to indicate that the following two fields reflect the result of the most recent export
SXx_SQ_Export_Position	SXx→SQ	1	Specifies whether there is room for another position.
SXx_SQ_Export_Buffer	SXx→SQ	7	Specifies the space available in the output buffers. 0: buffers are full 1: 2K-bits available (32-bits for each of the 64 pixels in a clause) ... 64: 128K-bits available (16 128-bit entries for each of 64 pixels) 65-127: RESERVED

~~25.1.16 *Shader Engine to Fetch Unit Bus*~~

Formatted: Bullets and Numbering

~~Four quad's worth of addresses is transferred to Fetch Unit every clock. These are sourced from a different pixel within each of the sub-engines repeating every 4 clocks. The register file index to read must precede the data by 2 clocks. The Read address associated with Quad 0 must be sent 1 clock after the Instruction Start signal is sent, so that data is read 3 clocks after the Instruction Start.~~

~~Four Quad's worth of Fetch Data may be written to the Register file every clock. These are directed to a different pixel of the sub-engines repeating every 4 clocks. The register file index to write must accompany the data. Data and Index associated with the Quad 0 must be sent 3 clocks after the Instruction Start signal is sent.~~

~~25.1.17~~25.1.11 *~~Sequencer to Fetch Unit bus~~SQ to TP: Control bus*

Formatted: Bullets and Numbering

Once every clock, the fetch unit sends to the sequencer on which clause it is now working and if the data in the ~~registers~~GPRs is ready or not. This way the sequencer can update the fetch counters for the reservation station fifos. The sequencer also provides the instruction and constants for the fetch to execute and the address in the register file where to write the fetch return data.

Name	Direction	Bits	Description
TPx_SQ_data_rdy	TPx→ SQ	1	Data ready

TPx_SQ_clause_num	TPx→ SQ	3	Clause number
SQ_TPx_const	SQ→TPx	64	Fetch state sent over 4 clocks
SQ_TPx_instuct	SQ→TPx	24	Fetch instruction sent over 4 clocks
SQ_TPx_end_of_clause	SQ→TPx	1	Last instruction of the clause
SQ_TPx_phase	SQ→TPx	2	Write phase signal
SQ_TP0_lod_correct	SQ→TP0	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP0_pmask	SQ→TP0	4	Pixel mask 1 bit per pixel
SQ_TP1_lod_correct	SQ→TP1	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP1_pmask	SQ→TP1	4	Pixel mask 1 bit per pixel
SQ_TP2_lod_correct	SQ→TP2	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP2_pmask	SQ→TP2	4	Pixel mask 1 bit per pixel
SQ_TP3_lod_correct	SQ→TP3	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP3_pmask	SQ→TP3	4	Pixel mask 1 bit per pixel
SQ_TPx_clause_num	SQ→TPx	3	Clause number
SQ_TPx_write_gpr_index	SQ->TPx	7	Index into Register file for write of returned Fetch Data

25.1.12 *TP to SQ: Texture stall*

Formatted: Bullets and Numbering

The TP sends this signal to the SQ when its input buffer is full. The SQ is going to send it to the SP X clocks after reception (maximum of 3 clocks of pipeline delay).

Name	Direction	Bits	Description
TP_SQ_fetch_stall	TP→ SQ	1	Do not send more texture request if asserted

25.1.13 *SQ to SP: Texture stall*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SPx_fetch_stall	SQ→SPx	1	Do not send more texture request if asserted

~~25.1.18~~25.1.14 *~~Sequencer~~Q to SP: GPR and Parameter cache control*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SPx~~_gpr~~_wr_addr	SQ→SPx	7	Write address
SQ_SPx_gpr_rd_addr	SQ→SPx	7	Read address
SQ_SPx_gpr_re_addr	SQ→SPx	1	Read Enable
SQ_SPx_gpr_we_addr	SQ→SPx	1	Write Enable for the GPRs
SQ_SPx_gpr_phase_mux	SQ→SPx	2	The phase mux
SQ_SPx_~~gpr_~~channel_mask	SQ→SPx	4	The channel mask
SQ_SP0_~~gpr_~~pixel_mask	SQ→SP0	4	The pixel mask
SQ_SP1_~~gpr_~~pixel_mask	SQ→SP1	4	The pixel mask
SQ_SP2_~~gpr_~~pixel_mask	SQ→SP2	4	The pixel mask
SQ_SP3_~~gpr_~~pixel_mask	SQ→SP3	4	The pixel mask
SQ_SPx_pc_we_addr	SQ→SPx	1	Write Enable for the parameter caches

~~25.1.19~~25.1.15 ~~Sequencer~~ SQ to SPx: Instructions

Formatted: Bullets and Numbering

Name ~~**Name**~~	**Direction**~~**Direction**~~	**Bits**~~**Bits**~~	**Description**~~**Description**~~
SQ_SPx_instruct_start~~SQ_SPx_instruct_start~~	SQ→SPx~~SQ→SPx~~	1~~1~~	Instruction start~~Instruction start~~
SQ_SP_instruct~~SQ_SP_instruct~~	SQ→SPx~~SQ→SPx~~	20~~20~~	Instruction sent over 4 clocks~~Instruction sent over 4 clocks~~
SQ_SPx_stall~~SQ_SPx_stall~~	SQ→SPx~~SQ→SPx~~	1~~1~~	Stall signal~~Stall signal~~
SQ_SPx_export_count~~SQ_SPx_Shader_Count~~	SQ→SPx~~SQ→SPx~~	3~~3~~	Each set of four pixels or vectors is exported over eight clocks. This field specifies where the SP is in that sequence.~~Each set of four pixels or vectors is exported over eight clocks. This field specifies where the SP is in that sequence.~~
SQ_SPx_export_last~~SQ_SPx_Shader_Last~~	SQ→SPx~~SQ→SPx~~	1~~1~~	Asserted on the first shader count of the last export of the clause~~Asserted on the first shader count of the last export of the clause~~
SQ_SP0_export_pvalid~~SQ_SP0_Shader_PixelValid~~	SQ→SP0~~SQ→SP0~~	4~~4~~	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock~~Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock~~
SQ_SP0_export_wvalid~~SQ_SP0_Shader_WordValid~~	SQ→SP0~~SQ→SP0~~	2~~2~~	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors~~Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors~~
SQ_SP1_export_pvalid~~SQ_SP1_Shader_PixelValid~~	SQ→SP1~~SQ→SP1~~	4~~4~~	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock~~Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock~~
SQ_SP1_	SQ→SP1~~SQ→SP1~~	2~~2~~	Specifies whether to write

export_wvalid~~SQ_SP1_Shader_WordValid~~			low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors~~Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors~~
SQ_SP2_ export_pvalid~~SQ_SP2_Shader_PixelValid~~	SQ→SP2~~SQ→SP2~~	4~~4~~	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock~~Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock~~
SQ_SP2_ export_wvalid~~SQ_SP2_Shader_WordValid~~	SQ→SP2~~SQ→SP2~~	2~~2~~	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors~~Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors~~
SQ_SP3_ export_pvalid~~SQ_SP3_Shader_PixelValid~~	SQ→SP3~~SQ→SP3~~	4~~4~~	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock~~Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock~~
SQ_SP3_ export_wvalid~~SQ_SP3_Shader_WordValid~~	SQ→SP3~~SQ→SP3~~	2~~2~~	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors~~Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors~~

~~25.1.20~~25.1.16 *SP to ~~Sequencer~~SQ: Constant address load*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SP0_SQ_const_addr	SP0→SQ	36	Constant address load to the sequencer
SP0_SQ_valid	SP0→SQ	1	Data valid
SP1_SQ_const_addr	SP1→SQ	36	Constant address load to the sequencer

SP1_SQ_valid	SP1→SQ	1	Data valid
SP2_SQ_const_addr	SP2→SQ	36	Constant address load to the sequencer
SP2_SQ_valid	SP2→SQ	1	Data valid
SP3_SQ_const_addr	SP3→SQ	36	Constant address load to the sequencer
SP3_SQ_valid	SP3→SQ	1	Data valid

~~25.1.21~~25.1.17 ~~*Sequencer*~~ *SQ to SPx: constant broadcast*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SPx_constant	SQ→SPx	128	Constant broadcast

~~25.1.22~~25.1.18 *SP0 to ~~Sequencer~~SQ: Kill vector load*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SP0_SQ_kill_vect	SP0→SQ	4	Kill vector load
SP1_SQ_kill_vect	SP1→SQ	4	Kill vector load
SP2_SQ_kill_vect	SP2→SQ	4	Kill vector load
SP3_SQ_kill_vect	SP3→SQ	4	Kill vector load

~~25.1.23~~25.1.19 *SQ to CP: RBBM bus*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_RBB_rs	SQ→CP	1	Read Strobe
SQ_RBB_rd	SQ→CP	32	Read Data
SQ_RBBM_nrtrtr	SQ→CP	1	Optional
SQ_RBBM_rtr	SQ→CP	1	Real-Time (Optional)

~~25.1.24~~25.1.20 *CP to SQ: RBBM bus*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
rbbm_we	CP→SQ	1	Write Enable
rbbm_a	CP→SQ	18	Address -- Upper Extent is TBD
rbbm_wd	CP→SQ	32	Data
rbbm_be	CP→SQ	4	Byte Enables
rbbm_re	CP→SQ	1	Read Enable
rbb_rs0	CP→SQ	1	Read Return Strobe 0
rbb_rs1	CP→SQ	1	Read Return Strobe 1
rbb_rd0	CP→SQ	32	Read Data 0
rbb_rd1	CP→SQ	32	Read Data 0
RBBM_SQ_soft_reset	CP→SQ	1	Soft Reset

26. Examples of program executions

26.1.1 *Sequencer Control of a Vector of Vertices*

1. PA sends a vector of 64 vertices (actually vertex indices – 32 bits/index for 2048 bit total) to the RE's Vertex FIFO
   - state pointer as well as tag into position cache is sent along with vertices
   - space was allocated in the position cache for transformed position before the vector was sent
   - **also before the vector is sent to the RE, the CP has loaded the global instruction store with the vertex shader program (using the MH?)**
   - The vertex program is assumed to be loaded when we receive the vertex vector.
     - the SEQ then accesses the IS base for this shader using the local state pointer (provided to all sequencers by the RBBM when the CP is done loading the program)
2. SEQ arbitrates between the Pixel FIFO and the Vertex FIFO – basically the Vertex FIFO always has priority
   - at this point the vector is removed from the Vertex FIFO
   - the arbiter is not going to select a vector to be transformed if the parameter cache is full unless the pipe as nothing else to do (ie no pixels are in the pixel fifo).

3. SEQ allocates space in the SP register file for index data plus GPRs used by the program
   - the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer that came down with the vertices
   - SEQ will not send vertex data until space in the register file has been allocated
4. SEQ sends the vector to the SP register file over the RE_SP interface (which has a bandwidth of 2048 bits/cycle)
   - the 64 vertex indices are sent to the 64 register files over 4 cycles
     - RF0 of SU0, SU1, SU2, and SU3 is written the first cycle
     - RF1 of SU0, SU1, SU2, and SU3 is written the second cycle
     - RF2 of SU0, SU1, SU2, and SU3 is written the third cycle
     - RF3 of SU0, SU1, SU2, and SU3 is written the fourth cycle
   - the index is written to the least significant 32 bits **(floating point format?) (what about compound indices)** of the 128-bit location within the register file (w); the remaining data bits are set to zero (x, y, z)
5. SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of fetch state machine 0, or TSM0 FIFO)
   - the control packet contains the state pointer, the tag to the position cache and a register file base pointer.
6. TSM0 accepts the control packet and fetches the instructions for fetch clause 0 from the global instruction store
   - TSM0 was first selected by the TSM arbiter before it could start
7. all instructions of fetch clause 0 are issued by TSM0
8. the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)
   - TSM0 does not wait for requests made to the Fetch Unit to complete; it passes the register file write index for the fetch data to the TU, which will write the data to the RF as it is received
   - once the TU has written all the data to the register files, it increments a counter that is associated with ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go ahead start to execute the ALU clause
9. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store
10. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of fetch state machine 1, or TSM1 FIFO)
11. the control packet continues to travel down the path of reservation stations until all clauses have been executed
    - position can be exported in ALU clause 3 (or 4?); the data (and the tag) is sent over a position bus (which is shared with all four shader pipes) back to the PA's position cache
    - A parameter cache pointer is also sent along with the position data. This tells to the PA where the data is going to be in the parameter cache.
      - there is a position export FIFO in the SP that buffers position data before it gets sent back to the PA
      - the ASM arbiter will prevent a packet from starting an exporting clause if the position export FIFO is full
    - parameter data is exported in clause 7 (as well as position data if it was not exported earlier)
      - parameter data is sent to the Parameter Cache over a dedicated bus
      - the SEQ allocates storage in the Parameter Cache, and the SEQ deallocates that space when there is no longer a need for the parameters (it is told by the PA when using a token).
      - the ASM arbiter will prevent a packet from starting on ASM7 if the parameter cache (or the position buffer if position is being exported) is full
12. after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program

26.1.2 *Sequencer Control of a Vector of Pixels*

1. **As with vertex shader programs, pixel shaders are loaded into the global instruction store by the CP**

   - At this point it is assumed that the pixel program is loaded into the instruction store and thus ready to be read.

2. the RE's Pixel FIFO is loaded with the barycentric coordinates for pixel quads by the detailed walker
   - the state pointer and the LOD correction bits are also placed in the Pixel FIF0
   - the Pixel FIFO is wide enough to source four quad's worth of barycentrics per cycle

3. SEQ arbitrates between Pixel FIFO and Vertex FIFO – when there are no vertices pending OR there is no space left in the register files for vertices, the Pixel FIFO is selected
4. SEQ allocates space in the SP register file for all the GPRs used by the program
   - the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer
   - SEQ will not allow interpolated data to be sent to the shader until space in the register file has been allocated
5. SEQ controls the transfer of interpolated data to the SP register file over the RE_SP interface (which has a bandwidth of 2048 bits/cycle). See interpolated data bus diagrams for details.
6. SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of fetch state machine 0, or TSM0 FIFO)
   - note that there is a separate set of reservation stations/arbiters/state machines for vertices and for pixels
   - the control packet contains the state pointer, the register file base pointer, and the LOD correction bits
   - all other information (such as quad address for example) travels in a separate FIFO
7. TSM0 accepts the control packet and fetches the instructions for fetch clause 0 from the global instruction store
   - TSM0 was first selected by the TSM arbiter before it could start
8. all instructions of fetch clause 0 are issued by TSM0
9. the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)
   - TSM0 does not wait for fetch requests made to the Fetch Unit to complete; it passes the register file write index for the fetch data to the TU, which will write the data to the RF as it is received
   - once the TU has written all the data for a particular clause to the register files, it increments a counter that is associated with the ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go ahead and pop the FIFO and start to execute the ALU clause
10. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store
11. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of fetch state machine 1, or TSM1 FIFO)
12. the control packet continues to travel down the path of reservation stations until all clauses have been executed
    - pixel data is exported in the last ALU clause (clause 7)
      - it is sent to an output FIFO where it will be picked up by the render backend
      - the ASM arbiter will prevent a packet from starting on ASM7 if the output FIFO is full
13. after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program

26.1.3 *Notes*

14. The state machines and arbiters will operate ahead of time so that they will be able to immediately start the real threads or stall.
15. The register file base pointer for a vector needs to travel with the vector through the reservation stations, but the instruction store base pointer does not – this is because the RF pointer is different for all threads, but the IS pointer is only different for each state and thus can be accessed via the state pointer.

~~16. Waterfalling still needs to be specked out.~~

Formatted: Bullets and Numbering

27. Open issues

~~There is currently an issue with constants. If the constants are not the same for the whole vector of vertices, we don't have the bandwidth from the fetch store to feed the ALUs. Two solutions exists for this problem:~~

~~1)Let the compiler handle the case and put those instructions in a fetch clause so we can use the bandwidth there to operate. This requires a significant amount of temporary storage in the register store.~~

Formatted: Bullets and Numbering

~~2)Waterfall down the pipe allowing only at a given time the vertices having the same constants to operate in parallel. This might in the worst case slow us down by a factor of 16.~~

Need to do some testing on the size of the register file as well as on the register file allocation method (dynamic VS static).

Saving power?

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 2015~~7~~ ~~January, 200211~~	GEN-CXXXXX-REVA	1 of 47

Author: Laurent Lefebvre

Issue To:	**Copy No:**

R400 Sequencer Specification

SQ

Version 1.6~~5~~

Overview: This is an architectural specification for the R400 Sequencer block (SEQ). It provides an overview of the required capabilities and expected uses of the block. It also describes the block interfaces, internal sub-blocks, and provides internal state diagrams.

AUTOMATICALLY UPDATED FIELDS:

Document Location: C:\perforce\r400\doc_lib\design\blocks\sq\R400_Sequencer.doc

Current Intranet Search Title: R400 Sequencer Specification

APPROVALS	
Name/Dept	Signature/Date

Remarks:

THIS DOCUMENT CO~~NTA~~read_ptrINS CONFIDENTIAL INFORMATION THAT COULD BE SUBSTANTIALLY DETRIME~~NTA~~read_ptrL TO THE INTEREST OF ATI TECHNOLOGIES INC. THROUGH UNAUTHORIZED USE OR DISCLOSURE.

Table Of Contents

1. OVERVIEW 6
1.1 Top Level Block Diagram 8
1.2 Data Flow graph 10
1.3 Control Graph 11
2. INTERPOLATED DATA BUS 11
3. INSTRUCTION STORE 14
4. SEQUENCER INSTRUCTIONS 16
5. CONSTANT STORES 16
5.1 Memory organizations 16
5.2 Management of the re-mapping tables 16
5.2.1 Dirty bits .... 19~~18~~
5.2.2 Free List Block .... 19~~18~~
5.2.3 De-allocate Block .... 19~~18~~
5.2.4 Operation of Incremental model .... 20~~18~~
5.3 Constant Store Indexing .... 21~~19~~
5.4 Real Time Commands .... 21~~19~~
6. LOOPING AND BRANCHES .... 22~~20~~
6.1 The controlling state. .... 22~~20~~
6.2 The Control Flow Program .... 22~~20~~
6.3 Data dependant predicate instructions .... 25~~23~~
6.4 HW Detection of PV,PS .... 25~~23~~
6.5 Register file indexing .... 26~~23~~
6.6 Predicated Instruction support for Texture clauses .... 26~~24~~
6.7 Debugging the Shaders .... 26~~24~~
6.7.1 Method 1: Debugging registers .... 26~~24~~
6.7.2 Method 2: Exporting the values in the GPRs (12) .... 27~~25~~
7. PIXEL KILL MASK .... 27~~25~~
8. MULTIPASS VERTEX SHADERS (HOS) .... 27~~25~~
9. REGISTER FILE ALLOCATION .... 27~~25~~
10. FETCH ARBITRATION .... 28~~26~~
11. ALU ARBITRATION .... 28~~26~~
12. HANDLING STALLS .... 29~~27~~
13. CONTENT OF THE RESERVATION STATION FIFOS .... 29~~27~~
14. THE OUTPUT FILE .... 29~~27~~
15. IJ FORMAT .... 29~~27~~
15.1 Interpolation of constant attributes .... 30~~28~~
16. THE PARAMETER CACHE .... 32~~3028~~
17. VERTEX POSITION EXPORTING .... 32~~3028~~
18. EXPORTING ARBITRATION .... 32~~3028~~
19. EXPORT TYPES .... 32~~3029~~
19.1 Vertex Shading .... 33~~3029~~
19.2 Pixel Shading .... 33~~3029~~
20. SPECIAL INTERPOLATION MODES .... 33~~3129~~

20.1 Real time commands .... 33~~3129~~
20.2 Sprites/ XY screen coordinates/ FB information .... 33~~3130~~
20.3 Auto generated counters .... 34~~3130~~
20.3.1 Vertex shaders .... 34~~3230~~
20.3.2 Pixel shaders .... 34~~3230~~
21. STATE MANAGEMENT .... 35~~3231~~
21.1 Parameter cache synchronization .... 35~~3231~~
22. XY ADDRESS IMPORTS .... 35~~3231~~
22.1 Vertex indexes imports .... 35~~3331~~
23. REGISTERS .... 35~~3331~~
23.1 Control .... 35~~3331~~
23.2 Context .... 36~~3332~~
24. DEBUG REGISTERS .... 36~~3433~~
24.1 Context .... 36~~3433~~
25. INTERFACES .... 37~~3433~~
25.1 External Interfaces .... 37~~3433~~
25.1.1 SC to SQ : IJ Control bus .... 37~~3433~~
25.1.2 SQ to SP: Interpolator bus .... 37~~3534~~
25.1.3 SQ to SP: GPR Input Mux select .... 38~~3534~~
25.1.4 SQ to SP: Parameter Cache Read control bus .... 38~~3635~~
25.1.5 SQ to SX: Parameter Cache Mux control Bus .... 38~~3635~~
25.1.6 SQ to SP: Staging Register Data .... 38~~3635~~
25.1.7 PA to SQ : Vertex interface .... 38~~3635~~
25.1.8 SQ to CP: State report .... 42~~4039~~
25.1.9 SQ to SX: Control bus .... 42~~4039~~
25.1.10 SX to SQ : Output file control .... 42~~4039~~
25.1.11 SQ to TP: Control bus .... 42~~4039~~
25.1.12 TP to SQ: Texture stall .... 43~~4140~~
25.1.13 SQ to SP: Texture stall .... 43~~4140~~
25.1.14 SQ to SP: GPR and Parameter cache control .... 43~~4140~~
25.1.15 SQ to SPx: Instructions .... 44~~4241~~
25.1.16 SP to SQ: Constant address load .... 44~~4241~~
25.1.17 SQ to SPx: constant broadcast .... 44~~4241~~
25.1.18 SP0 to SQ: Kill vector load .... 45~~4342~~
25.1.19 SQ to CP: RBBM bus .... 45~~4342~~
25.1.20 CP to SQ: RBBM bus .... 45~~4342~~
26. EXAMPLES OF PROGRAM EXECUTIONS .... 45~~4342~~
26.1.1 Sequencer Control of a Vector of Vertices .... 45~~4342~~
26.1.2 Sequencer Control of a Vector of Pixels .... 46~~4443~~
26.1.3 Notes .... 47~~4544~~

27. OPEN ISSUES..47~~4544~~

Revision Changes:

Rev 0.1 (Laurent Lefebvre) Date: May 7, 2001	First draft.
Rev 0.2 (Laurent Lefebvre) Date : July 9, 2001	Changed the interfaces to reflect the changes in the SP. Added some details in the arbitration section.
Rev 0.3 (Laurent Lefebvre) Date : August 6, 2001	Reviewed the Sequencer spec after the meeting on August 3, 2001.
Rev 0.4 (Laurent Lefebvre) Date : August 24, 2001	Added the dynamic allocation method for register file and an example (written in part by Vic) of the flow of pixels/vertices in the sequencer.
Rev 0.5 (Laurent Lefebvre) Date : September 7, 2001	Added timing diagrams (Vic)
Rev 0.6 (Laurent Lefebvre) Date : September 24, 2001	Changed the spec to reflect the new R400 architecture. Added interfaces.
Rev 0.7 (Laurent Lefebvre) Date : October 5, 2001	Added constant store management, instruction store management, control flow management and data dependant predication.
Rev 0.8 (Laurent Lefebvre) Date : October 8, 2001	Changed the control flow method to be more flexible. Also updated the external interfaces.
Rev 0.9 (Laurent Lefebvre) Date : October 17, 2001	Incorporated changes made in the 10/18/01 control flow meeting. Added a NOP instruction, removed the conditional_execute_or_jump. Added debug registers.
Rev 1.0 (Laurent Lefebvre) Date : October 19, 2001	Refined interfaces to RB. Added state registers.
Rev 1.1 (Laurent Lefebvre) Date : October 26, 2001	Added SEQ→SP0 interfaces. Changed delta precision. Changed VGT→SP0 interface. Debug Methods added.
Rev 1.2 (Laurent Lefebvre) Date : November 16, 2001	Interfaces greatly refined. Cleaned up the spec.
Rev 1.3 (Laurent Lefebvre) Date : November 26, 2001	Added the different interpolation modes.
Rev 1.4 (Laurent Lefebvre) Date : December 6, 2001	Added the auto incrementing counters. Changed the VGT→SQ interface. Added content on constant management. Updated GPRs.
Rev 1.5 (Laurent Lefebvre) Date : December 11, 2001	Removed from the spec all interfaces that weren't directly tied to the SQ. Added explanations on constant management. Added PA→SQ synchronization fields and explanation.
Rev 1.6 (Laurent Lefebvre) Date : January 7, 2002	Added more details on the staging register. Added detail about the parameter caches. Changed the call instruction to a Conditionnal call instruction. Added details on constant management and updated the diagram.

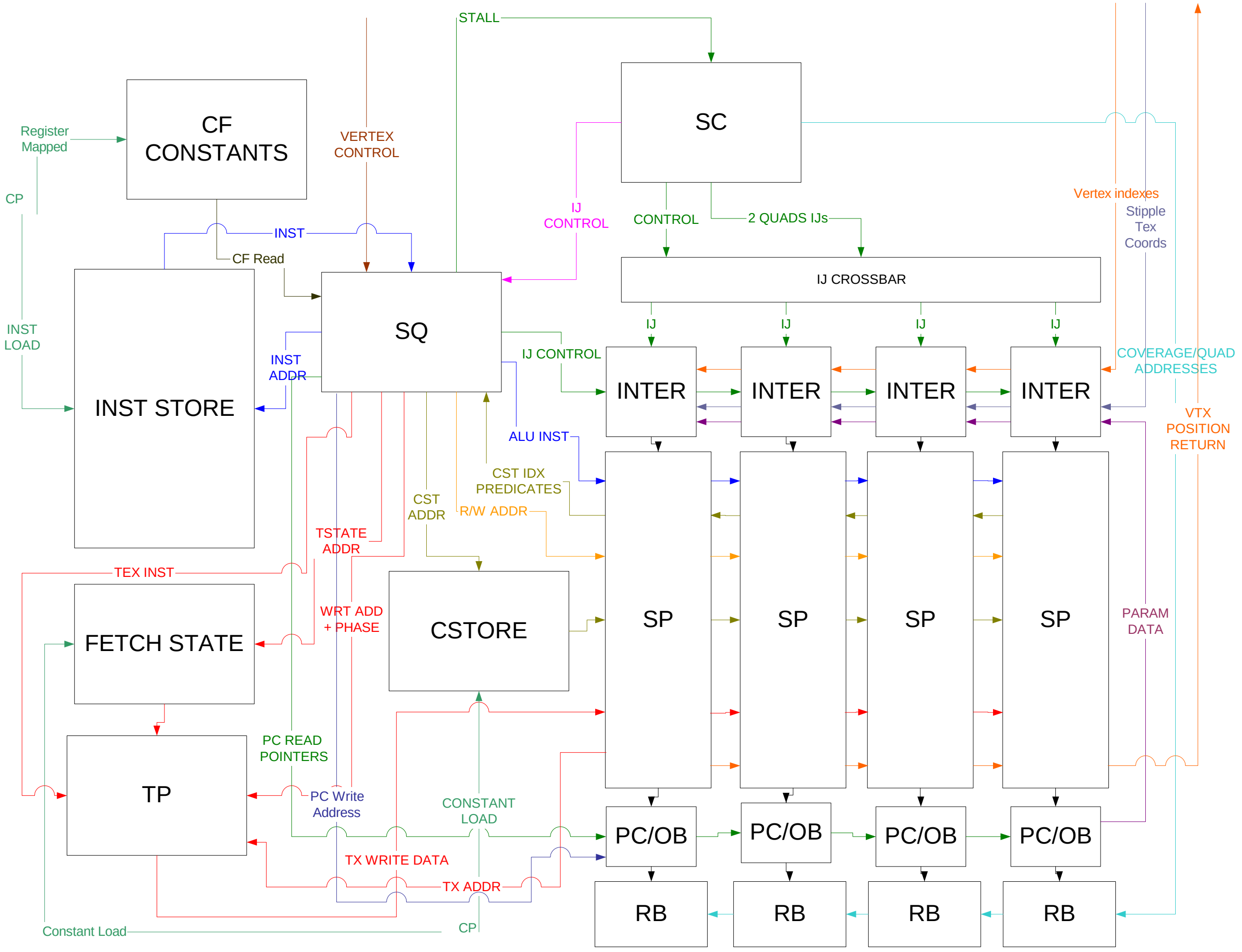
1. Overview

The sequencer is based on the R300 design. It chooses two ALU clauses and a fetch clause to execute, and executes all of the instructions in a clause before looking for a new clause of the same type. Two ALU clauses are executed interleaved to hide the ALU latency. Each vector will have eight fetch and eight ALU clauses, but clauses do not need to contain instructions. A vector of pixels or vertices ping-pongs along the sequencer FIFO, bouncing from fetch reservation station to alu reservation station. A FIFO exists between each reservation stage, holding up vectors until the vector currently occupying a reservation station has left. A vector at a reservation station can be chosen to execute. The sequencer looks at all eight alu reservation stations to choose an alu clause to execute and all eight fetch stations to choose a fetch clause to execute. The arbitrator will give priority to clauses/reservation stations closer to the bottom of the pipeline. It will not execute an alu clause until the fetch fetches initiated by the previous fetch clause have completed. There are two separate sets of reservation stations, one for pixel vectors and one for vertices vectors. This way a pixel can pass a vertex and a vertex can pass a pixel.

To support the shader pipe the sequencer also contains the shader instruction cache, constant store, control flow constants and texture state. The four shader pipes also execute the same instruction thus there is only one sequencer for the whole chip.

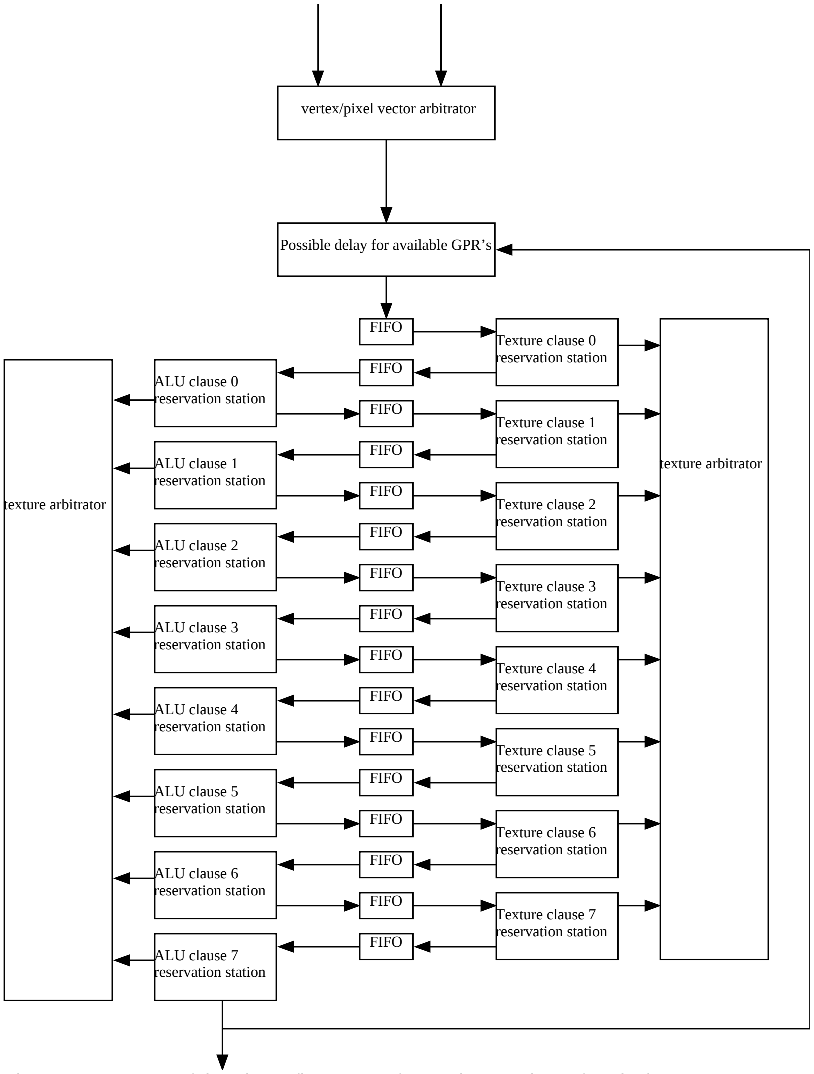
The sequencer first arbitrates between vectors of 64 vertices that arrive directly from primitive assembly and vectors of 16 quads (64 pixels) that are generated in the scan converter.

The vertex or pixel program specifies how many GPRs it needs to execute. The sequencer will not start the next vector until the needed space is available in the GPRs.



STALL
SC
CF
CONSTANTS
Register
Mapped
CP
VERTEX
CONTROL
IJ
CONTROL
CONTROL
2 QUADS IJs
Vertex indexes
Stipple
Tex
Coords
INST
CF Read
SQ
IJ CROSSBAR
IJ
IJ
IJ
IJ
INST
LOAD
INST STORE
INST
ADDR
IJ CONTROL
INTER
INTER
INTER
INTER
COVERAGE/QUAD
ADDRESSES
VTX
POSITION
RETURN
ALU INST
CST IDX
PREDICATES
CST
ADDR
R/W ADDR
TSTATE
ADDR
SP
SP
SP
SP
TEX INST
WRT ADD
+ PHASE
CSTORE
FETCH STATE
PARAM
DATA
PC READ
POINTERS
TP
PC Write
Address
CONSTANT
LOAD
PC/OB
PC/OB
PC/OB
PC/OB
TX WRITE DATA
TX ADDR
RB
RB
RB
RB
Constant Load
CP

1.1 Top Level Block Diagram


vertex/pixel vector arbitrator
Possible delay for available GPR's
FIFO
Texture clause 0 reservation station
ALU clause 0 reservation station
Texture clause 1 reservation station
ALU clause 1 reservation station
Texture clause 2 reservation station
ALU clause 2 reservation station
Texture clause 3 reservation station
ALU clause 3 reservation station
Texture clause 4 reservation station
ALU clause 4 reservation station
Texture clause 5 reservation station
ALU clause 5 reservation station
Texture clause 6 reservation station
ALU clause 6 reservation station
Texture clause 7 reservation station
ALU clause 7 reservation station
texture arbitrator
texture arbitrator

There are two sets of the above figure, one for vertices and one for pixels.

Depending on the arbitration state, the sequencer will either choose a vertex or a pixel packet. The control packet consists of 3 bits of state, 7 bits for the base address of the Shader program and some information on the coverage to determine fetch LOD plus other various small state bits.

On receipt of a packet, the input state machine (not pictured but just before the first FIFO) allocated enough space in the GPRs to store the interpolated values and temporaries. Following this, the barycentric coordinates (and XY screen position if needed) are sent to the interpolator which will use them to interpolate the parameters and place the results into the GPRs. Then, the input state machine stacks the packet in the first FIFO.

On receipt of a command, the level 0 fetch machine issues a fetch request to the TP and corresponding GPR address for the fetch address (ta). A small command (tcmd) is passed to the fetch system identifying the current level number (0) as well as the GPR write address for the fetch return data. One fetch request is sent every 4 clocks causing the texturing of sixteen 2x2s worth of data (or 64 vertices). Once all the requests are sent the packet is put in FIFO 1.

Upon receipt of the return data, the fetch unit writes the data to the register file using the write address that was provided by the level 0 fetch machine and sends the clause number (0) to the level 0 fetch state machine to signify that the write is done and thus the data is ready. Then, the level 0 fetch machine increments the counter of FIFO 1 to signify to the ALU 0 that the data is ready to be processed.

On receipt of a command, the level 0 ALU machine first decrements the input FIFO 1 counter and then issues a complete set of level 0 shader instructions. For each instruction, the ALU state machine generates 3 source addresses, one destination address and an instruction. Once the last instruction has been issued, the packet is put into FIFO 2.

There will always be two active ALU clauses at any given time (and two arbiters). One arbiter will arbitrate over the odd instructions (4 clocks cycles) and the other one will arbitrate over the even instructions (4 clocks cycles). The only constraints between the two arbiters is that they are not allowed to pick the same clause number as the other one is currently working on if the packet is not of the same type (render state).

If the packet is a vertex packet, upon reaching ALU clause 3, it can export the position if the position is ready. So the arbiter must prevent ALU clause 3 to be selected if the positional buffer is full (or can't be accessed). Along with the positional data, if needed the sprite size and/or edge flags can also be sent.

~~{ISSUE: How do we handle parameter cache pointers (computed, semi-computed or not computed)?}~~

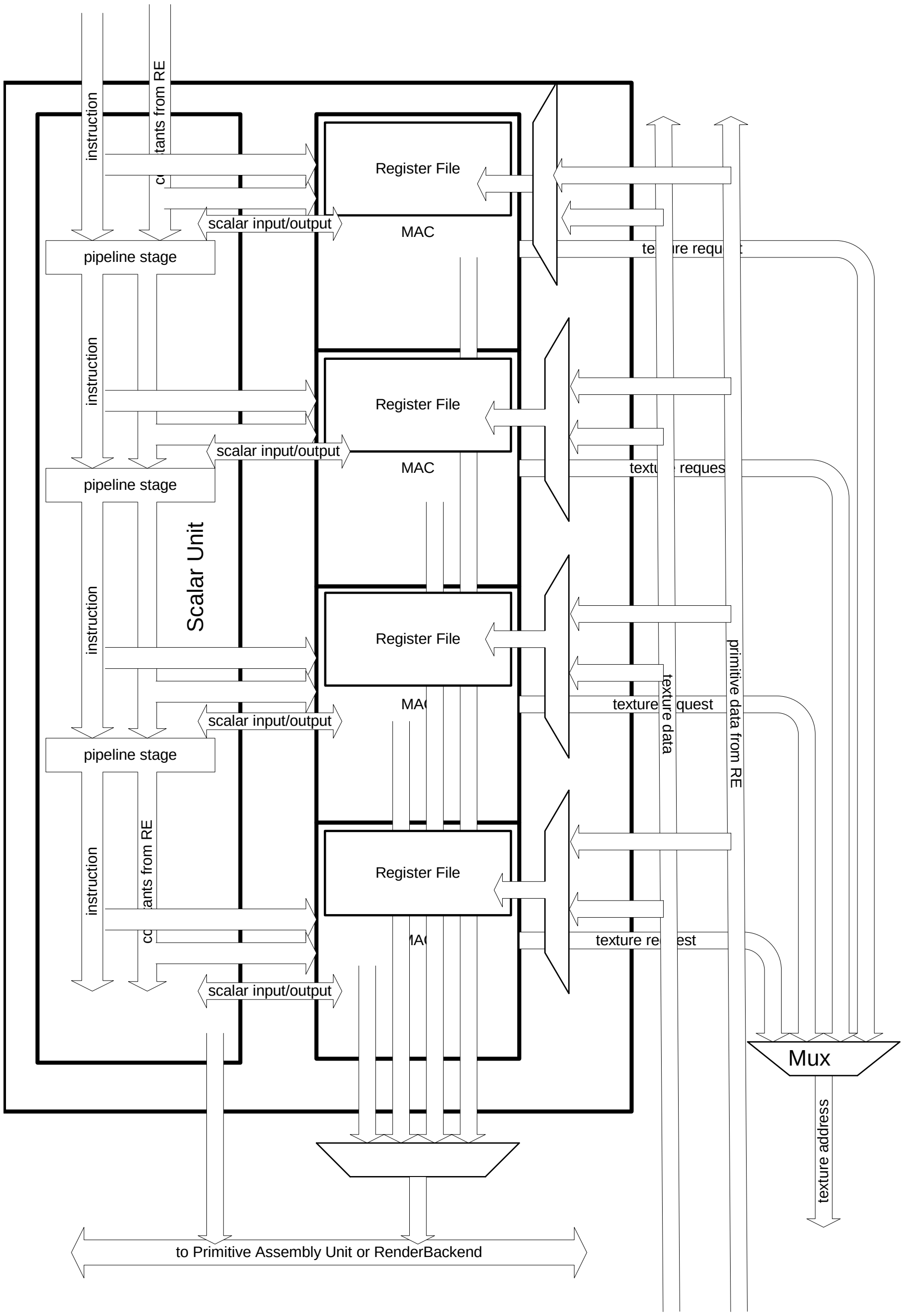
A special case is for multipass vertex shaders, which can export 12 parameters per last 6 clauses to the output buffer. If the output buffer is full or doesn't have enough space the sequencer will prevent such a vertex group to enter an exporting clause.

Multipass pixel shaders can export 12 parameters to memory from the last clause only (7).

All other clauses process in the same way until the packet finally reaches the last ALU machine (7).

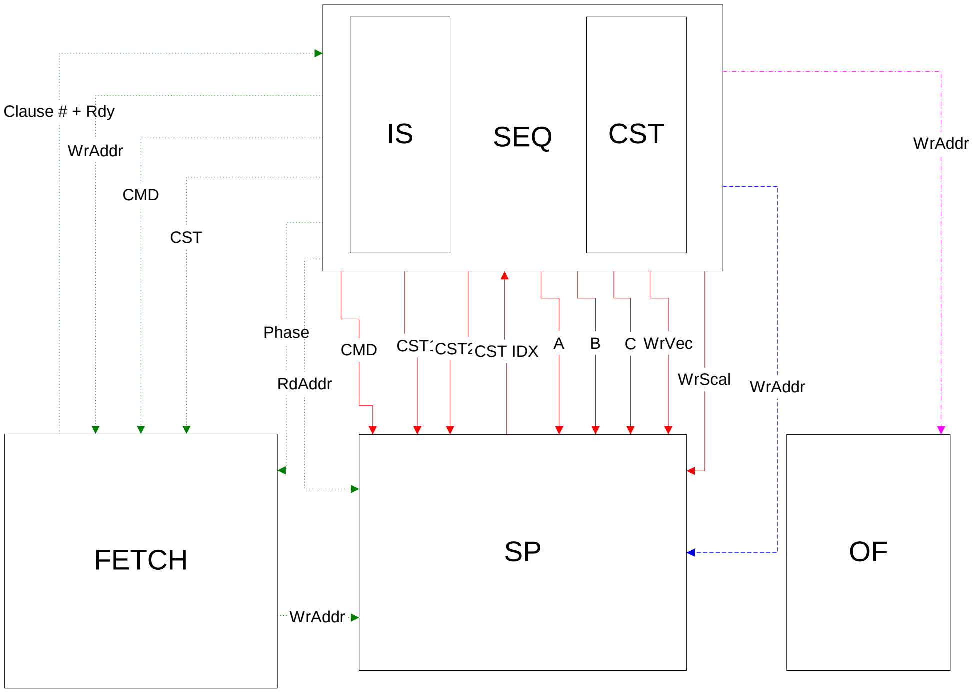
Only one pair of interleaved ALU state machines may have access to the register file address bus or the instruction decode bus at one time. Similarly, only one fetch state machine may have access to the register file address bus at one time. Arbitration is performed by three arbiter blocks (two for the ALU state machines and one for the fetch state machines). The arbiters always favor the higher number state machines, preventing a bunch of half finished jobs from clogging up the register files.

1.2 Data Flow graph (SP)



The gray area represents blocks that are replicated 4 times per shader pipe (16 times on the overall chip).

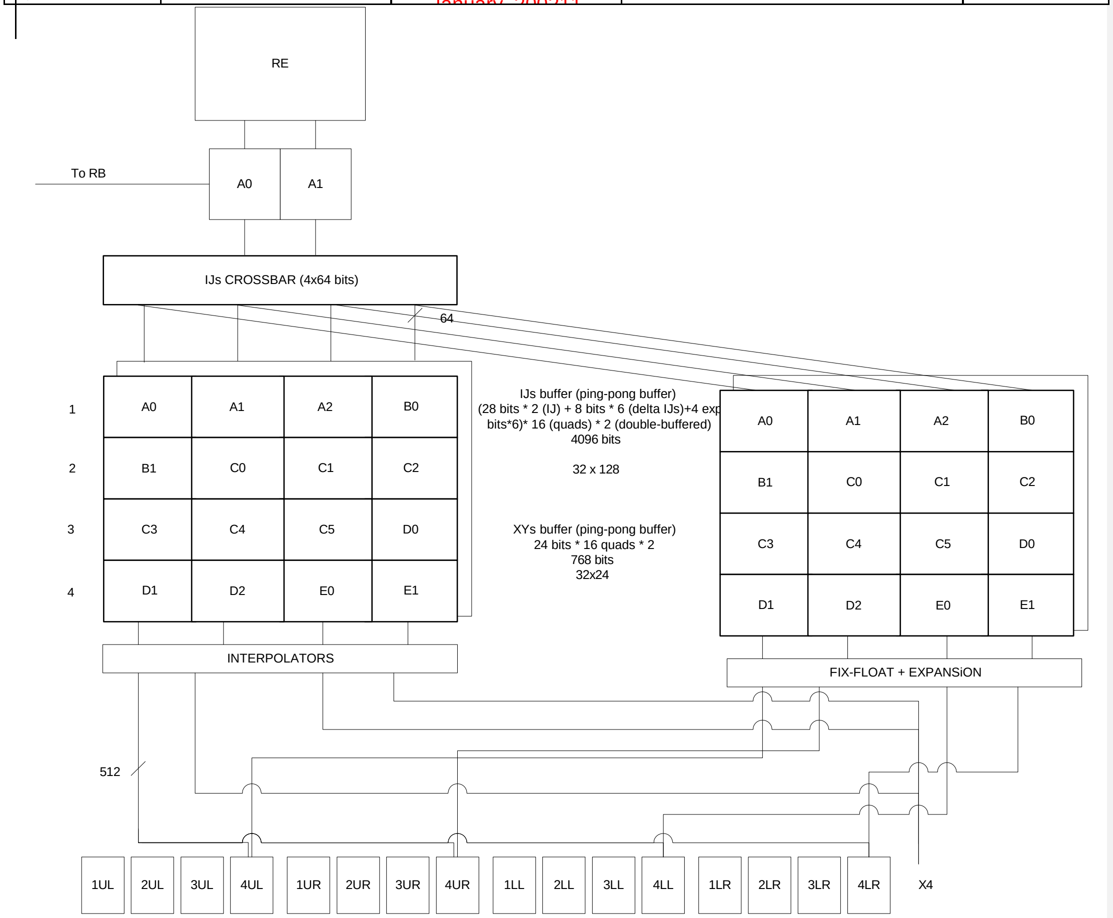
1.3 Control Graph



In green is represented the Fetch control interface, in red the ALU control interface, in blue the Interpolated/Vector control interface and in purple is the output file control interface.

2. Interpolated data bus

The interpolators contain an IJ buffer to pack the information as much as possible before writing it to the register file.



RE
To RB
A0
A1
IJs CROSSBAR (4x64 bits)
64
1
2
3
4
A0
A1
A2
B0
B1
C0
C1
C2
C3
C4
C5
D0
D1
D2
E0
E1
IJs buffer (ping-pong buffer)
(28 bits * 2 (IJ) + 8 bits * 6 (delta IJs)+4 exp
bits*6)* 16 (quads) * 2 (double-buffered)
4096 bits
32 x 128
XYs buffer (ping-pong buffer)
24 bits * 16 quads * 2
768 bits
32x24
A0
A1
A2
B0
B1
C0
C1
C2
C3
C4
C5
D0
D1
D2
E0
E1
INTERPOLATORS
FIX-FLOAT + EXPANSiON
512
1UL
2UL
3UL
4UL
1UR
2UR
3UR
4UR
1LL
2LL
3LL
4LL
1LR
2LR
3LR
4LR
X4

WRITES

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21	T22	T23
SP 0	A0	A0	XY A0	B1	B1	XY B1	C3	C3	XY C3				D1	D1	XY D1									
SP 1	A1	A1	XY A1				C0	C0	XY C0	C4	C4	XY C4	D2	D2	XY D2									
SP 2	A2	A2	XY A2				C1	C1	XY C1	C5	C5	XY C5				E0	E0	XY E0						
SP 3				B0	B0	XY B0	C2	C2	XY C2				D0	D0	XY D0	E1	E1	XY E1						

READS

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21	T22	T23
SP 0	XY 0-3	XY 16-19	XY 32-35	XY 48-51	A0	B1	C3	D1					A0	B1	C3	D1					V 0-3	V 16-19	V 32-35	V 48-51
SP 1	XY 4-7	XY 20-23	XY 36-39	XY 52-55	A1		C4	D2		C0			A1		C4	D2		C0			V 4-7	V 20-23	V 36-39	V 52-55
SP 2	XY 8-11	XY 24-27	XY 40-43	XY 56-59	A2		C5			C1		E0	A2		C5			C1		E0	V 8-11	V 24-27	V 40-43	V 56-59
SP 3	XY 12-15	XY 28-31	XY 44-47	XY 60-63					B0	C2	D0	E1					B0	C2	D0	E1	V 12-15	V 28-31	V 44-47	V 60-63
	XY				P1								P2								VTX			

Above is an example of a tile the sequencer might receive from the SC. The write side is how the data get stacked into the XY and IJ buffers, the read side is how the data is passed to the GPRs. The IJ information is packed in the IJ buffer 4 quads at a time or two clocks. The sequencer allows at any given time as many as four quads to interpolate a parameter. They all have to come from the same primitive. Then the sequencer controls the write mask to the GPRs to write the valid data in.

3. Instruction Store

There is going to be only one instruction store for the whole chip. It will contain 4096 instructions of 96 bits each.

It is likely to be a 1 port memory; we use 1 clock to load the ALU instruction, 1 clocks to load the Fetch instruction, 1 clock to load 2 control flow instructions and 1 clock to write instructions.

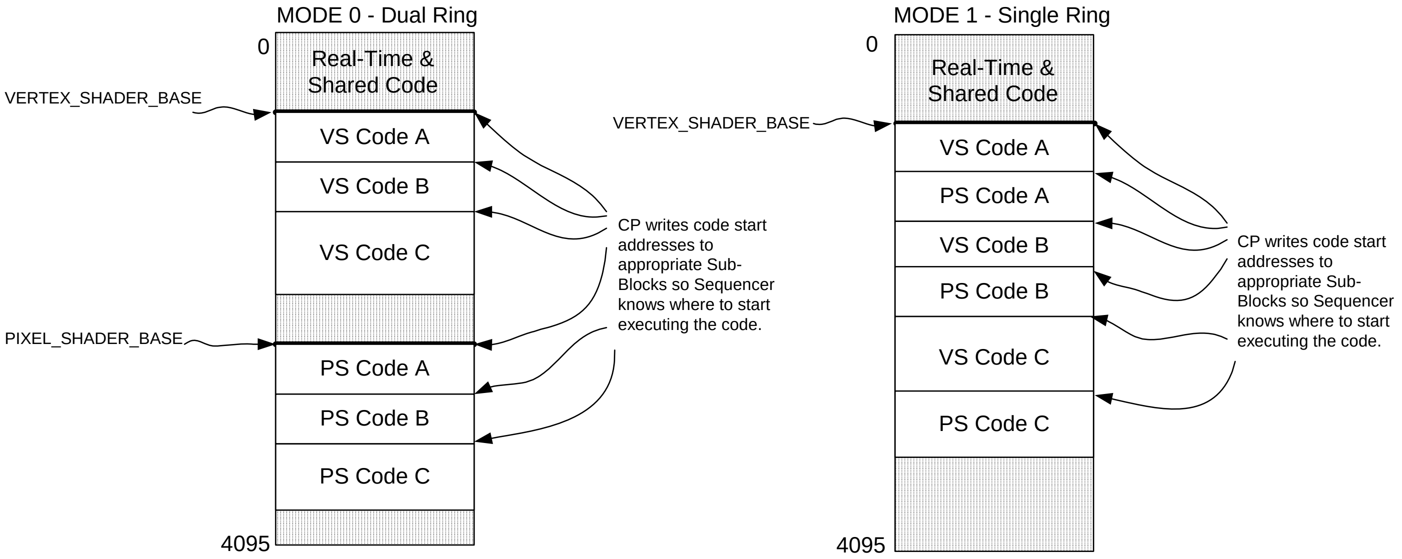
The instruction store is loaded by the CP thru the ~~INST_DATA, INST_INDEX_PORT control register. The INST_INDEX_PORT is auto-incremented on both reads and writes to the INST_DATA register.~~register mapped registers.

The next picture shows the various modes the CP can load the memory. The Sequencer has to keep track of the loading modes in order to wrap around the correct boundaries. ~~The MSB of the INST_INDEX_PORT register contains the packet type for the sequencer to know where it must wrap around.~~ The ~~wrap around points are~~wrap around points are arbitrary and they are specified in the VS_BASE and PIX_BASE registers.

For the Real time commands the story is quite the same but for some small differences. ~~The CP will use the INST_INDEX_PORT_RT and INST_DATA_RT register pair instead of the regular ones and T~~there are no wrap around points for real time so the driver must be careful not to overwrite regular shader data. The shared code (shared subroutines) uses the same path as real time.

R400 CP's Views of Instruction Memory

Updated: 11/14/2001
John A. Carey



4. Sequencer Instructions

All control flow instructions and move instructions are handled by the sequencer only. The ALUs will perform NOPs during this time (MOV PV,PV, PS,PS) if they have nothing else to do.

5. Constant Stores

5.1 Memory organizations

A likely size for the ALU constant store is 1024x128 bits. The read BW from the ALU constant store is 128 bits/clock and the write bandwidth is 32 bits/clock (directed by the CP bus size not by memory ports).

The maximum logical size of the constant store for a given shader is 256 constants. Or 512 for the pixel/vertex shader pair. The size of the re-mapping table is 128 lines (each line addresses 4 constants). The write granularity is 4 constants or 512 bits. It takes 16 clocks to write the four constants.

The texture state is also kept in a similar memory. The size of this memory is 192x128. The memory thus holds 128 texture states (192 bits per state). The logical size exposes 32 different states total, which are going to be shared between the pixel and the vertex shader. The size of the re-mapping table to for the texture state memory is 16 lines (each line addresses 2 texture state lines in the real memory). The write granularity is 2 texture state lines (or 384 bits). The driver sends 512 bits but the CP ignores the top 128 bits. It thus takes 12 clocks to write the two texture states.

The control flow constant memory doesn’t sit behind a renaming table. It is register mapped and thus the driver must reload its content each time there is a state change. Its size is 320*32 because it must hold 8 copies of the 32 dwords of control flow constants and the loop construct constants must be aligned.

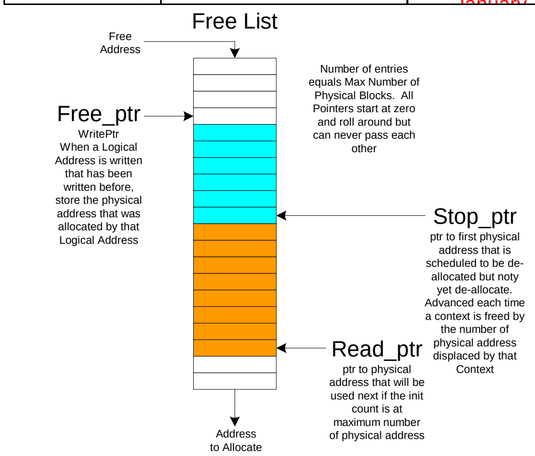
~~The CP is loading the constant store using the CONST_DATA and CONST_ADDR registers. It does so by writing to the CONST_ADDR register the logical address for the constant block it wants to update and then writes 16 times to the CONST_DATA register. The CONST_ADDR is auto-incremented on both reads and writes to the CONST_DATA register.~~ The constant re-mapping tables for texture state and ALU constants are logically register mapped for regular mode and physically register mapped for RT operation.

5.2 Management of the re-mapping tables

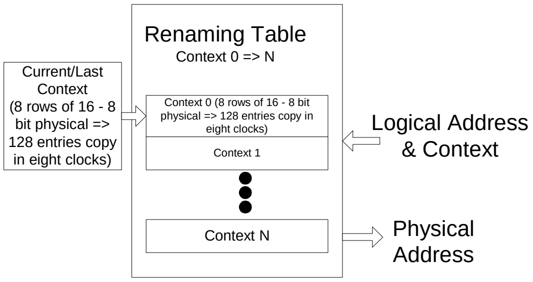
The sequencer is responsible to manage two re-mapping tables (one for the constant store and one for the texture state). On a state change (by the driver), the sequencer will broadside copy the contents of its re-mapping tables to a new one. We have 8 different re-mapping tables we can use concurrently.

The constant memory update will be incremental, the driver only need to update the constants that actually changed between the two state changes.

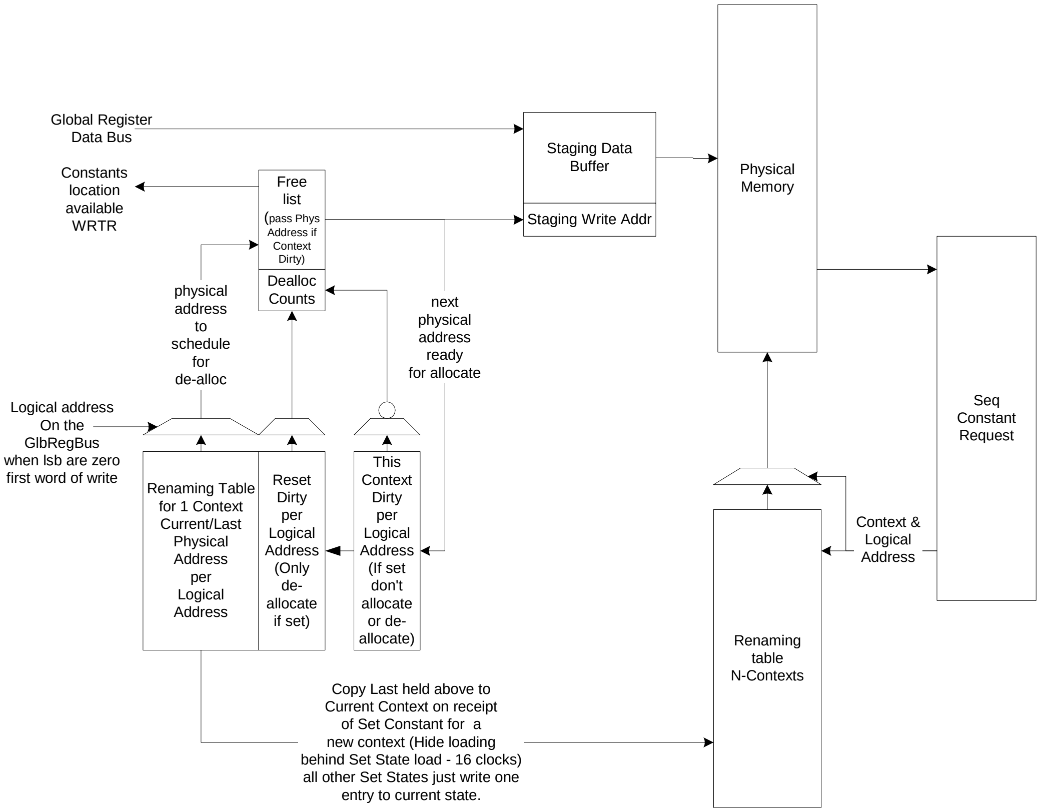
For this model to work, the requirement is that the physical memory MUST be at least twice as large as the logical address space. In our case, since the logical address space is 512, the memory must be of sizes 1024 and above.



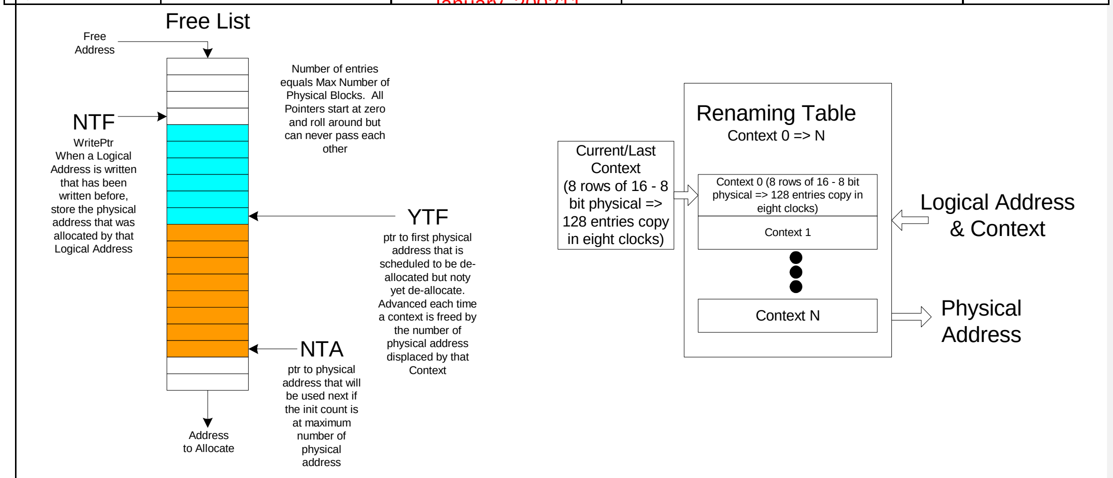
Free List
Free Address
Free_ptr
WritePtr
When a Logical Address is written that has been written before, store the physical address that was allocated by that Logical Address
Number of entries equals Max Number of Physical Blocks. All Pointers start at zero and roll around but can never pass each other
Stop_ptr
ptr to first physical address that is scheduled to be de-allocated but noty yet de-allocate. Advanced each time a context is freed by the number of physical address displaced by that Context
Read_ptr
ptr to physical address that will be used next if the init count is at maximum number of physical address
Address to Allocate

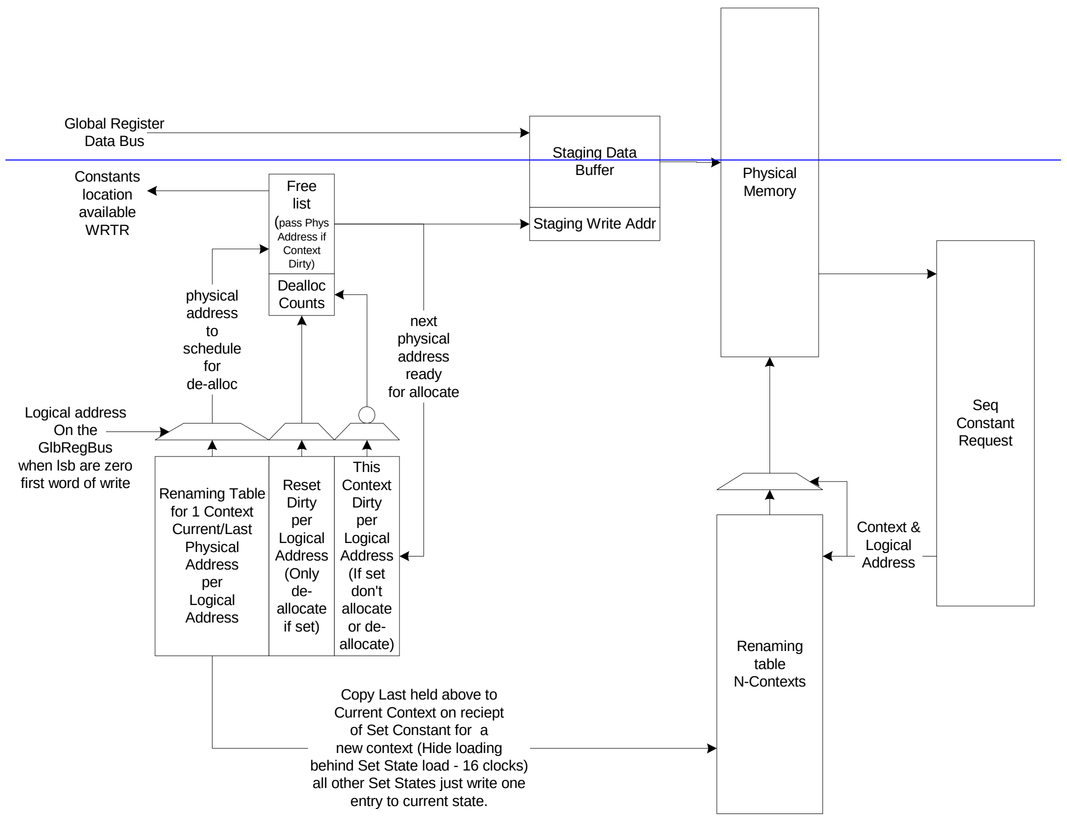


Current/Last Context (8 rows of 16 - 8 bit physical => 128 entries copy in eight clocks)
Renaming Table
Context 0 => N
Context 0 (8 rows of 16 - 8 bit physical => 128 entries copy in eight clocks)
Context 1
Context N
Logical Address & Context
Physical Address



Global Register Data Bus
Staging Data Buffer
Staging Write Addr
Physical Memory
Constants location available WRTR
Free list (pass Phys Address if Context Dirty)
Dealloc Counts
physical address to schedule for de-alloc
next physical address ready for allocate
Logical address On the GlbRegBus when lsb are zero first word of write
Renaming Table for 1 Context Current/Last Physical Address per Logical Address
Reset Dirty per Logical Address (Only de-allocate if set)
This Context Dirty per Logical Address (If set don't allocate or de-allocate)
Seq Constant Request
Context & Logical Address
Renaming table N-Contexts
Copy Last held above to Current Context on receipt of Set Constant for a new context (Hide loading behind Set State load - 16 clocks) all other Set States just write one entry to current state.


Free List
Free Address
NTF
WritePtr When a Logical Address is written that has been written before, store the physical address that was allocated by that Logical Address
Number of entries equals Max Number of Physical Blocks. All Pointers start at zero and roll around but can never pass each other
YTF
ptr to first physical address that is scheduled to be de-allocated but noty yet de-allocate. Advanced each time a context is freed by the number of physical address displaced by that Context
NTA
ptr to physical address that will be used next if the init count is at maximum number of physical address
Address to Allocate
Current/Last Context (8 rows of 16 - 8 bit physical => 128 entries copy in eight clocks)
Renaming Table
Context 0 => N
Context 0 (8 rows of 16 - 8 bit physical => 128 entries copy in eight clocks)
Context 1
Context N
Logical Address & Context
Physical Address


Global Register Data Bus
Staging Data Buffer
Staging Write Addr
Physical Memory
Constants location available WRTR
Free list (pass Phys Address if Context Dirty)
Dealloc Counts
physical address to schedule for de-alloc
next physical address ready for allocate
Logical address On the GlbRegBus when lsb are zero first word of write
Renaming Table for 1 Context Current/Last Physical Address per Logical Address
Reset Dirty per Logical Address (Only de-allocate if set)
This Context Dirty per Logical Address (If set don't allocate or de-allocate)
Seq Constant Request
Context & Logical Address
Renaming table N-Contexts
Copy Last held above to Current Context on reciept of Set Constant for a new context (Hide loading behind Set State load - 16 clocks) all other Set States just write one entry to current state.

5.2.1 *Dirty bits*

Two sets of dirty bits will be maintained per logical address. The first one will be set to zero on reset and set when the logical address is addressed. The second one will be set to zero when ever a new context is written and set for each address written while in this context. The reset dirty is not set, then writing to that logical address will not require de-allocation of whatever address stored in the renaming table. If it is set and the context dirty is not set, then the physical address store needs to be de-allocated and a new physical address is necessary to store the incoming data. If they are both set, then the data will be written into the physical address held in the renaming for the current logical address. No de-allocation or allocation takes place. This will happen when the driver does a set constant twice to the same logical address between context changes. NOTE: It is important to detect and prevent this, failure to do it will allow multiple writes to allocate all physical memory and thus hang because a context will not fit for rendering to start and thus free up space.

5.2.2 *Free List Block*

A free list block that would consist of a counter (called the IFC or Initial Free Counter) that would reset to zero and incremented every time a chunk of physical memory is used until they have all been used once. This counter would be checked each time a physical block is needed, and if the original ones have not been used up, us a new one, else check the free list for an available physical block address. The count is the physical address for when getting a chunk from the counter.

Storage of a free list big enough to store all physical block addresses.

Maintain three pointers for the free list that are reset to zero. The first one we will call write_ptr. This pointer will identify the next location to write the physical address of a block to be de-allocated. Note: we can never free more physical memory locations than we have. Once recording address the pointer will be incremented to walk the free list like a ring.

The second pointer will be called stop_ptr. The stop_ptr pointer will be advanced by the number of address chunks de-allocates when a context finishes. The address between the stop_ptr and write_ptr cannot be reused because they are still in use. But as soon as the context using then is dismissed the stop_ptr will be advanced.

The third pointer will be called read_ptr. This pointer will point will point to the next address that can be used for allocation as long as the read_ptr does not equal the stop_ptr and the IFC is at its maximum count.

~~A free list block that would consist of a counter (called the IFC or Initial Free Counter) that would reset to zero and incremented every time a chunk of physical memory is used until they have all been used once. This counter would be checked each time a physical block is needed, and if the original ones have not been used up, us a new one, else check the free list for an available physical block address. The count is the physical address for when getting a chunk from the counter.~~

~~Storage of a free list big enough to store all physical block addresses.~~

~~Maintain three pointers for the free list that are reset to zero. The first one we will call NTF (Next To Free). This pointer will identify the next location to write the physical address of a block to be de-allocated. Note: we can never free more physical memory locations than we have. Once recording address the pointer will be incremented to walk the free list like a ring.~~

~~The second pointer will be called YTF (Yet To Free). The YTF pointer will be advanced by the number of address chunks de-allocates when a context finishes. The address between the YTF and NTF cannot be reused because they are still in use. But as soon as the context using then is dismissed the YTF will be advanced.~~

~~The third pointer will be called NTA (Next To Allocate). This pointer will point will point to the next address that can be used for allocation as long as the NTA does not equal the YTF and the IFC is at its maximum count.~~

5.2.3 *De-allocate Block*

This block will maintain a free physical address block count for each context. While in current context, a count shall be maintained specifying how many blocks were written into the free list at the write_ptr pointer. This count will be reset upon reset or when this context is active on the back and different than the previous context. It is actually a count of blocks in the previous context that will no longer be used. This count will be used to advance the write_ptr pointer to make available the set of physical blocks freed when the previous context was done. This allows the discard or de-allocation of any number of blocks in one clock.

~~This block will maintain a free physical address block count for each context. While in current context, a count shall be maintained specifying how many blocks were written into the free list at the NTF pointer. This count will be reset upon reset or when this context is active on the back and different than the previous context. It is actually a count of blocks in the previous context that will no longer be used. This count will be used to advance the NTF pointer to make available the set of physical blocks freed when the previous context was done. This allows the discard or de-allocation of any number of blocks in one clock.~~

5.2.4 *Operation of Incremental model*

The basic operation of the model would start with the write_ptr, stop_ptr, read_ptr pointers in the free list set to zero and the free list counter is set to zero. Also all the dirty bits and the previous context will be initialized to zero. When the first set constants happen, the reset dirty bit will not be set, so we will allocate a physical location from the free list counter because its not at the max value. The data will be written into physical address zero. Both the additional copy of the renaming table and the context zeros of the big renaming table will be updated for the logical address that was written by set start with physical address of 0. This process will be repeated for any logical address that are not dirty until the context changes. If a logical address is hit that has its dirty bits set while in the same context, both dirty bits would be set, so the new data will be over-written to the last physical address assigned for this logical address. When the first draw command of the context is detected, the previous context stored in the additional renaming table will be copied to the larger renaming table in the current (new) context location. Then the set constant logical address with be loaded with a new physical address during the copy and if the reset dirty was set, the physical address it replaced in the renaming table would be entered at the write_ptr pointer location on the free list and the write_ptr will be incremented. The de-allocation counter for the previous context (eight) will be incremented. This as set states come in for this context one of the following will happen:

1.) No dirty bits are set for the logical address being updated. A line will be allocated of the free-list counter or the free list at read_ptr pointer if read_ptr != to stop_ptr .
2.) Reset dirty set and Context dirty not set. A new physical address is allocated, the physical address in the renaming table is put on the free list at write_ptr and it is incremented along with the de-allocate counter for the last context.
3.) Context dirty is set then the data will be written into the physical address specified by the logical address.

Formatted: Bullets and Numbering

This process will continue as long as set states arrive. This block will provide backpressure to the CP whenever he has not free list entries available (counter at max and stop_ptr == read_ptr). The command stream will keep a count of contexts of constants in use and prevent more than max constants contexts from being sent.

Now preferable when the constant context leaves the last ALU clause it will be sent to this block and compared with the previous context that left. (Init to zero) If they differ than the older context will no longer be referenced and thus can be de-allocated in the physical memory. This is accomplished by adding the number of blocks freed this context to the stop_ptr pointer. This will make all the physical addresses used by this context available to the read_ptr allocate pointer for future allocation.

This device allows representation of multiple contexts of constants data with N copies of the logical address space. It also allows the second context to be represented as the first set plus some new additional data by just storing the delta's. It allows memory to be efficiently used and when the constants updates are small it can store multiple context. However, if the updates are large, less contexts will be stored and potentially performance will be degraded. Although it will still perform as well as a ring could in this case.~~The basic operation of the model would start with the NTF, YTF, NTA pointers in the free list set to zero and the free list counter is set to zero. Also all the dirty bits and the previous context will be initialized to zero. When the first set constants happen, the reset dirty bit will not be set, so we will allocate a physical location from the free list counter because its not at the max value. The data will be written into physical address zero. Both the additional copy of the renaming table and the context zeros of the big renaming table will be updated for the logical address that was written by set start with physical address of 0. This process will be repeated for any logical address that are not dirty until the context changes. If a logical address is hit that has its dirty bits set while in the same context, both dirty bits would be set, so the new data will be over-written to the last physical address assigned for this logical address. When a set constant comes with a different than last context, the previous context stored in the additional renaming table will be copied to the larger renaming table in the current (new) context location. Then the set constant logical address with be loaded with a new physical address during the copy and if the reset dirty was set, the physical address it replaced in the renaming table would be entered at the NTF pointer location on the free list and the NTF will be incremented. The de-allocation counter for the previous context (zero) will be incremented. This as set states come in for this context one of the following will happen:~~

~~1.)No dirty bits are set for the logical address being updated. A line will be allocated of the free-list counter or the free list at NTA pointer if NTA != to YTF .~~
~~2.)Reset dirty set and Context dirty not set. A new physical address is allocated, the physical address in the renaming table is put on the free list at NTF and it is incremented along with the de-allocate counter for the last context.~~
~~3.)Context dirty is set then the data will be written into the physical address specified by the logical address.~~

Formatted: Bullets and Numbering

~~This process will continue as long as set states arrive. This block will provide back pressure to the CP when ever he has not free list entries available (counter at max and YTF == NTA). The command stream will keep a count of contexts of constants in use and prevent more than max constants contexts from being sent.~~
~~Now preferable when the constant context leaves the last ALU clause it will be sent to this block and compared with the previous context that left. (Init to zero) If they differ than the older context will no longer be referenced and thus can be de-allocated in the physical memory. This is accomplished by adding the number of blocks freed this context to the YTF pointer. This will make all the physical addresses used by this context available to the NTA allocate pointer for future allocation.~~

~~This device allows representation of multiple contexts of constants data with N copies of the logical address space. It also allows the second context to be represented as the first set plus some new additional data by just storing the delta's. It allows memory to be efficiently used and when the constants updates are small it can store multiple context. However, if the updates are large, less contexts will be stored and potentially performance will be degraded. Although it will still perform as well as a ring could in this case.~~

5.3 Constant Store Indexing

In order to do constant store indexing, the sequencer must be loaded first with the indexes (that come from the GPRs). There are 144 wires from the exit of the SP to the sequencer (9 bits pointers x 16 vertexes/clock). Since the data must pass thru the Shader pipe for the float to fixed conversion, there is a latency of 4 clocks (1 instruction) between the time the sequencer is loaded and the time one can index into the constant store. The assembly will look like this

```
MOVA  R1.X,R2.X       // Loads the sequencer with the content of R2.X, also copies the content of R2.X into R1.X
NOP                   // latency of the float to fixed conversion
ADD   R3,R4,C0[R2.X]// Uses the state from the sequencer to add R4 to C0[R2.X] into R3
```

Note that we don't really care about what is in the brackets because we use the state from the MOVA instruction. R2.X is just written again for the sake of simplicity and coherency.

The storage needed in the sequencer in order to support this feature is 2*64*9 bits = 1152 bits.

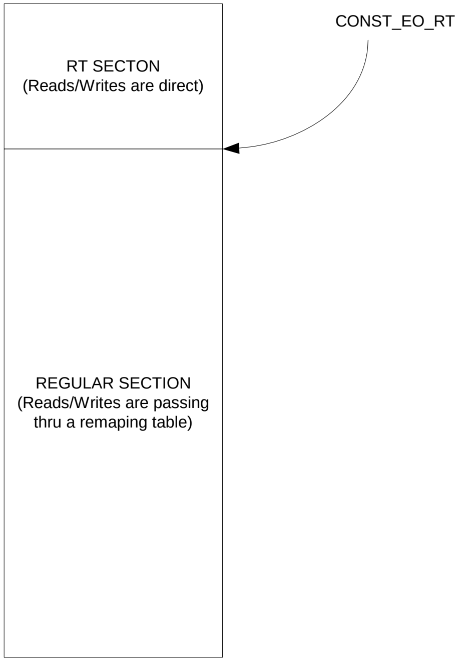
5.4 Real Time Commands

The real time commands constants are written by the CP using the ~~CONST_DATA_RT and CONST_ADDR_RT GPRs~~register mapped registers allocated for RT. It works is the same way than when dealing with regular constant loads BUT in this case the CP is not sending a logical address but rather a physical address and the reads are not passing thru the re-mapping table but are directly read from the memory. The boundary between the two zones is defined by the CONST_EO_RT control register.

5.5 Constant Waterfalling

Formatted: Bullets and Numbering

In order to have a reasonable performance in the case of constant store indexing using the address register, we are going to have the possibility of using the physical memory port for read only. This way we can read 1 constant per clock and thus have a worst-case waterfall mode of 1 vertex per clock. There is a small synchronization issue related with this as we need for the SQ to make sure that the constants where actually written to memory (not only sent to the sequencer) before it can allow the first vector of pixels or vertices of the state to go thru the ALUs. To do so, the sequencer keeps 8 bits (one per render state) and sets the bits whenever the last render state is written to memory and clears the bit whenever a state is freed.



CONST_EO_RT
RT SECTON
(Reads/Writes are direct)
REGULAR SECTION
(Reads/Writes are passing thru a remaping table)

6. Looping and Branches

Loops and branches are planned to be supported and will have to be dealt with at the sequencer level. We plan on supporting constant loops and branches using a control program.

6.1 The controlling state.

The R400 controling state consists of:

Boolean[256:0]
Loop_count[7:0][31:0]
Loop_Start[7:0][31:0]
Loop_Step[7:0][31:0]

That is 256 Booleans and 32 loops.

We have a stack of 4 elements for nested calls of subroutines and 4 loop counters to allow for nested loops.

This state is available on a per shader program basis.

6.2 The Control Flow Program

Examples of control flow programs are located in the R400 programming guide document.

The basic model is as follows:

The render state defined the clause boundaries:
Vertex_shader_fetch[7:0][7:0]   // eight 8 bit pointers to the location where each clauses control program is located
Vertex_shader_alu[7:0][7:0]   // eight 8 bit pointers to the location where each clauses control program is located

Pixel_shader_fetch[7:0][7:0]   // eight 8 bit pointers to the location where each clauses control program is located
Pixel_shader_alu[7:0][7:0]   // eight 8 bit pointers to the location where each clauses control program is located

A pointer value of FF means that the clause doesn’t contain any instructions.

The control program for a given clause is executed to completion before moving to another clause, (with the exception of the pick two nature of the alu execution). The control program is the only program aware of the clause boundaries.

The control program has eleven basic instructions:

Execute
Conditional_execute
Conditional_Execute_Predicates
Conditional_jump
Conditionnal_Call
Return
Loop_start
Loop_end
End_of_clause
Conditional_End_of_clause
NOP

Execute, causes the specified number of instructions in instruction store to be executed.
Conditional_execute checks a condition first, and if true, causes the specified number of instructions in instruction store to be executed.
Loop_start resets the corresponding loop counter to the start value on the first pass after it checks for the end condition and if met jumps over to a specified address.
Loop_end increments (decrements?) the loop counter and jumps back the specified number of instructions.
Conditionnal_Call jumps to an address and pushes the IP counter on the stack if the condition is met. On the return instruction, the IP is popped from the stack.
Conditional_execute_or_Jump executes a block of instructions or jumps to an address is the condition is not met.
Conditional_execute_Predicates executes a block of instructions if all bits in the predicate vectors meet the condition.
End_of_clause marks the end of a clause.
Conditional_End_of_clause marks the end of a clause if the condition is met.
Conditional_jumps jumps to an address if the condition is met.
NOP is a regular NOP

NOTE THAT ALL JUMPS MUST JUMP TO EVEN CFP ADDRESSES since there are two control flow instructions per memory line. Thus the compiler must insert NOPs where needed to align the jumps on even CFP addresses.

Also if the jump is logically bigger than pshader_cntl_size (or vshader_cntl_size) we break the program (clause) and set the debug registers. If an execute or conditional_execute is lower than cntl_size or bigger than size we also break the program (clause) and set the debug registers.

We have to fit instructions into 48 bits in order to be able to put two control flow instruction per line in the instruction store.

Note that whenever a field is marked as RESERVED, it is assumed that all the bits of the field are cleared (0).

Execute				
47	46… 42	41 … 24	23 … 12	11 … 0
Addressing	00001	RESERVED	Instruction count	Exec Address

Execute up to 4k instructions at the specified address in the instruction memory.

NOP		
47	46 … 42	41 … 0

Addressing	00010	RESERVED

This is a regular NOP.

Conditional_Execute							
47	46 … 42	41	40 … 33	32	31 … 24	23 … 12	11 … 0
Addressing	00011	RESERVED	Boolean address	Condition	RESERVED	Instruction count	Exec Address

If the specified Boolean (8 bits can address 256 Booleans) meets the specified condition then execute the specified instructions (up to 4k instructions)

Conditional_Execute_Predicates							
47	46 … 42	41 … 35	34 … 33	32	31 … 24	23 … 12	11 … 0
Addressing	00100	RESERVED	Predicate vector	Condition	RESERVED	Instruction count	Exec Address

Check the AND/OR of all current predicate bits. If AND/OR matches the condition execute the specified number of instructions. We need to AND/OR this with the kill mask in order not to consider the pixels that aren't valid.

Loop_Start				
47	46 … 42	41 … 17	16 … 12	11 … 0
Addressing	00101	RESERVED	loop ID	Jump address

Loop Start. Compares the loop iterator with the end value. If loop condition not met jump to the address. Forward jump only. Also computes the index value. The loop id must match between the start to end, and also indicates which control flow constants should be used with the loop.

Loop_End				
47	46 … 42	41 … 17	16 … 12	11 … 0
Addressing	00111	RESERVED	loop ID	start address

Loop end. Increments the counter by one, compares the loop count with the end value. If loop condition met, continue, else, jump BACK to the start of the loop.

The way this is described does not prevent nested loops, and the inclusion of the loop id make this easy to do.

Conditionnal_Call						
47	46 … 42	41 … 35~~41…12~~	34 … 33	32	31 … 12	11 … 0
Addressing	01000	RESERVED	Predicate vector	Condition	RESERVED	Jump address

If the condition is met, j~~J~~umps to the specified address and pushes the control flow program counter on the stack.

Return		
47	46 … 42	41 … 0
Addressing	01001	RESERVED

Pops the topmost address from the stack and jumps to that address. If nothing is on the stack, the program will just continue to the next instruction.

Conditionnal_Jump							
47	46 … 42	41	40 … 33	32	31	30 … 12	11 … 0

Addressing	01010	RESERVED	Boolean address	Condition	FW only	RESERVED	Jump address

If condition met, jumps to the address. FORWARD jump only allowed if bit 31 set. Bit 31 is only an optimization for the compiler and should NOT be exposed to the API.

Conditional_End_of_Clause					
47	46 … 42	41	40 … 33	32	31 … 0
Addressing	01011	RESERVED	Boolean address	Condition	RESERVED

This is an optimization in the case of very short shaders (where the control flow instruction can't be hidden anymore and thus are not free. In this case, if the condition is met, the clause is ended, else we continue the execution of the clause.

End_of_Clause		
47	46 … 42	41 … 0
Addressing	01011	RESERVED

Marks the end of a clause.

To prevent infinite loops, we will keep 9 bits loop counters instead of 8 (we are only able to loop 256 times). If the counter goes higher than 255 then the loop_end or the loop_start instruction is going to break the loop and set the debug GPRs.

6.3 Data dependant predicate instructions

Data dependant conditionals will be supported in the R400. The only way we plan to support those is by supporting three vector/scalar predicate operations of the form:

PRED_SETE_# - similar to SETE except that the result is 'exported' to the sequencer.
PRED_SETNE_# - similar to SETNE except that the result is 'exported' to the sequencer.
PRED_SETGT_# - similar to SETGT except that the result is 'exported' to the sequencer
PRED_SETGTE_# - similar to SETGTE except that the result is 'exported' to the sequencer

For the scalar operations only we will also support the two following instructions:
PRED_SETE0_# – SETE0
PRED_SETE1_# – SETE1

The export is a single bit - 1 or 0 that is sent using the same data path as the MOVA instruction. The sequencer will maintain 4 sets of 64 bit predicate vectors (in fact 8 sets because we interleave two programs but only 4 will be exposed) and use it to control the write masking. This predicate is not maintained across clause boundaries. The # sign is used to specify which predicate set you want to use 0 thru 3.

Then we have two conditional execute bits. The first bit is a conditional execute "on" bit and the second bit tells us if we execute on 1 or 0. For example, the instruction:

P0_ADD_# R0,R1,R2

Is only going to write the result of the ADD into those GPRs whose predicate bit is 0. Alternatively, P1_ADD_# would only write the results to the GPRs whose predicate bit is set. The use of the P0 or P1 without precharging the sequencer with a PRED instruction is undefined.

{Issue: do we have to have a NOP between PRED and the first instruction that uses a predicate?}

6.4 HW Detection of PV,PS

Because of the control program, the compiler cannot detect statically dependant instructions. In the case of non-masked writes and subsequent reads the sequencer will insert uses of PV,PS as needed. This will be done by comparing the read address and the write address of consecutive instructions. For masked writes, the sequencer will insert NOPs wherever there is a dependant read/write.

The sequencer will also have to insert NOPs between PRED_SET and MOVA instructions and their uses.

6.5 Register file indexing

Because we can have loops in fetch clause, we need to be able to index into the register file in order to retrieve the data created in a fetch clause loop and use it into an ALU clause. The instruction will include the base address for register indexing and the instruction will contain these controls:

Bit7	Bit 6	
0	0	'absolute register'
0	1	'relative register'
1	0	'previous vector'
1	1	'previous scalar'

In the case of an absolute register we just take the address as is. In the case of a relative register read we take the base address and we add to it the loop_index and this becomes our new address that we give to the shader pipe.

The sequencer is going to keep a loop index computed as such:

Index = Loop_iterator*Loop_step + Loop_start.

The index is going to return 0 if it is out of the range. We loop until loop_iterator = loop_count. Loop_step is a signed value [-128…127].

6.6 Predicated Instruction support for Texture clauses

For texture clauses, we support the following optimization: we keep 1 bit (thus 4 bits for the four predicate vectors) per predicate vector in the reservation stations. A value of 1 means that one ore more elements in the vector have a value of one (thus we have to do the texture fetches for the whole vector). A value of 0 means that no elements in the vector have his predicate bit set and we can thus skip over the texture fetch. **We have to make sure the invalid pixels aren't considered with this optimization.**

6.7 Debugging the Shaders

In order to be able to debug the pixel/vertex shaders efficiently, we provide 2 methods.

6.7.1 *Method 1: Debugging registers*

Current plans are to expose 2 debugging, or error notification, registers:
1. address register where the first error occurred
2. count of the number of errors

The sequencer will detect the following groups of errors:
- count overflow
~~- jump error~~
~~-relative jump address > size of the control flow program~~
~~-relative jump address > length of the shader program~~
- ~~-~~constant overflow
-
- register overflow
~~- register overflow~~
~~- call stack~~
~~-call with stack full~~
~~-return with stack empty~~
Compiler recognizable errors:
- jump errors

Formatted: Bullets and Numbering

relative jump address > size of the control flow program
relative jump address > length of the shader program
- call stack
call with stack full
return with stack empty

With two of the errors, a jump error or a register overflow will cause the program to break. In this case, a break means that a clause will halt execution, but allowing further clauses to be executed.

With the other errors, program can continue to run, potentially to worst-case limits.

If indexing outside of the constant range, causing an overflow error, the hardware is specified to return the value with an index of 0. This could be exploited to generate error tokens, by reserving and initializing the 0th register (or constant) for errors.

{ISSUE : Interrupt to the driver or not?}

6.7.2 *Method 2: Exporting the values in the GPRs (12)*

The sequencer will have a count register and an address register for this mode and 3 bits per clause specifying the execution mode for each clause. The modes can be :

1) Normal
2) Debug Kill
3) Debug Addr + Count

Under the normal mode execution follows the normal course. Under the kill mode, all control flow instructions are executed but all normal shader instructions of the clause are replaced by NOPs. Only debug_export instructions of clause 7 will be executed under the debug kill setting. Under the other mode, normal execution is done until we reach an address specified by the address register and instruction count (useful for loops) specified by the count register. After we have hit the instruction n times (n=count) we switch the clause to the kill mode.

Under the debug mode (debug kill OR debug Addr + count), it is assumed that clause 7 is always exporting 12 debug vectors and that all other exports to the SX block (position, color, z, ect) will been turned off (changed into NOPs) by the sequencer (even if they occur before the address stated by the ADDR debug register).

7. Pixel Kill Mask

A vector of 64 bits is kept by the sequencer per group of pixels/vertices. Its purpose is to optimize the texture fetch requests and allow the shader pipe to kill pixels using the following instructions:

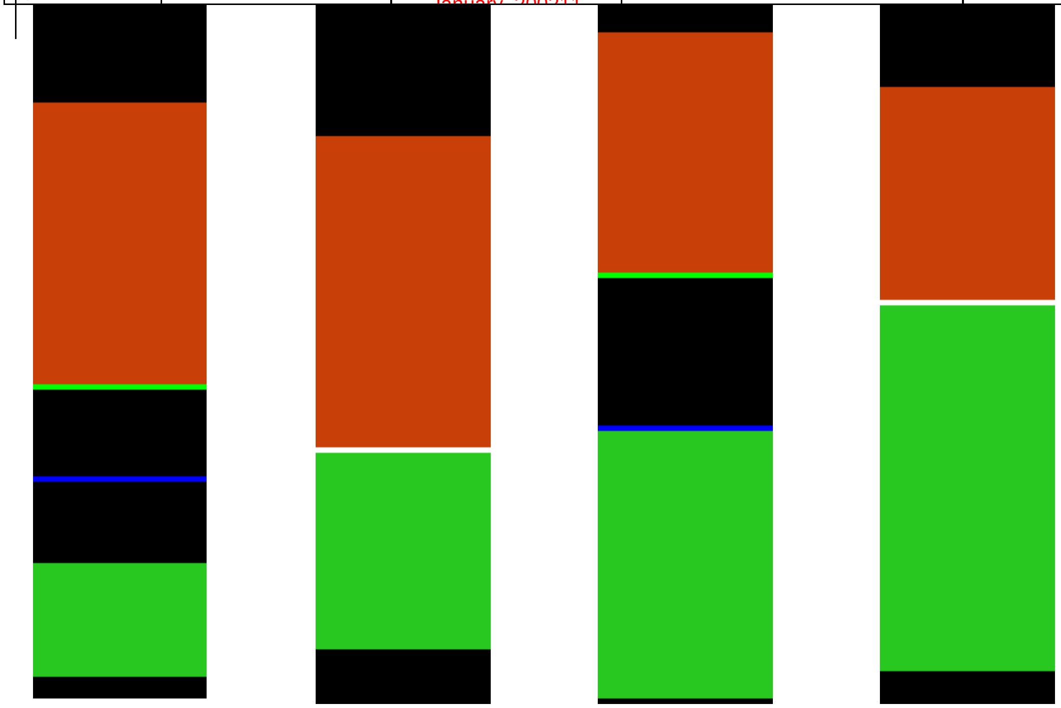
MASK_SETE
MASK_SETNE
MASK_SETGT
MASK_SETGTE

8. Multipass vertex shaders (HOS)

Multipass vertex shaders are able to export from the 6 last clauses but to memory ONLY.

9. Register file allocation

The register file allocation for vertices and pixels can either be static or dynamic. In both cases, the register file in managed using two round robins (one for pixels and one for vertices). In the dynamic case the boundary between pixels and vertices is allowed to move, in the static case it is fixed to VERTEX_REG_SIZE for vertices and 256-VERTEX_REG_SIZE for pixels.



Above is an example of how the algorithm works. Vertices come in from top to bottom; pixels come in from bottom to top. Vertices are in orange and pixels in green. The blue line is the tail of the vertices and the green line is the tail of the pixels. Thus anything between the two lines is shared. When pixels meets vertices the line turns white and the boundary is static until both vertices and pixels share the same “unallocated bubble”. Then the boundary is allowed to move again.

10. Fetch Arbitration

The fetch arbitration logic chooses one of the 8 potentially pending fetch clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute. Once chosen, the clause state machine will send one 2x2 fetch per clock (or 4 fetches in one clock every 4 clocks) until all the fetch instructions of the clause are sent. This means that there cannot be any dependencies between two fetches of the same clause.

The arbitrator will not wait for the fetches to return prior to selecting another clause for execution. The fetch pipe will be able to handle up to X(?) in flight fetches and thus there can be a fair number of active clauses waiting for their fetch return data.

11. ALU Arbitration

ALU arbitration proceeds in almost the same way than fetch arbitration. The ALU arbitration logic chooses one of the 8 potentially pending ALU clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute. There are two ALU arbiters, one for the even clocks and one for the odd clocks. For example, here is the sequencing of two interleaved ALU clauses (E and O stands for Even and Odd sets of 4 clocks):

Einst0 Oinst0 Einst1 Oinst1 Einst2 Oinst2 Einst0 Oinst3 Einst1 Oinst4 Einst2 Oinst0…
Proceeding this way hides the latency of 8 clocks of the ALUs. Also note that the interleaving also occurs across clause boundaries.

12. Handling Stalls

When the output file is full, the sequencer prevents the ALU arbitration logic from selecting the last clause (this way nothing can exit the shader pipe until there is place in the output file. If the packet is a vertex packet and the position buffer is full (POS_FULL) then the sequencer also prevents a thread from entering the exporting clause (3?). The sequencer will set the OUT_FILE_FULL signal n clocks before the output file is actually full and thus the ALU arbiter will be able read this signal and act accordingly by not preventing exporting clauses to proceed.

13. Content of the reservation station FIFOs

The reservation FIFOs contain the state of the vector of pixels and vertices. We have two sets of those: one for pixels, and one for vertices. They contain 3 bits of Render State 7 bits for the base address of the GPRs, some bits for LOD correction and coverage mask information in order to fetch fetch for only valid pixels, the quad address.

14. The Output File

The output file is where pixels are put before they go to the RBs. The write BW to this store is 256 bits/clock. Just before this output file are staging registers with write BW 512 bits/clock and read BW 256 bits/clock. The staging registers are 4x128 (and there are 16 of those on the whole chip).

15. IJ Format

The IJ information sent by the PA is of this format on a per quad basis:

We have a vector of IJ's (one IJ per pixel at the centroid of the fragment or at the center of the pixel depending on the mode bit). The interpolation is done at a different precision across the 2x2. The upper left pixel's parameters are always interpolated at full 20x24 mantissa precision. Then the result of the interpolation along with the difference in IJ in reduced precision is used to interpolate the parameter for the other three pixels of the 2x2. Here is how we do it:

Assuming P0 is the interpolated parameter at Pixel 0 having the barycentric coordinates I(0), J(0) and so on for P1,P2 and P3. Also assuming that A is the parameter value at V0 (interpolated with I), B is the parameter value at V1 (interpolated with J) and C is the parameter value at V2 (interpolated with (1-I-J).

$$\Delta 01I = I(1) - I(0)$$
$$\Delta 01J = J(1) - J(0)$$
$$\Delta 02I = I(2) - I(0)$$
$$\Delta 02J = J(2) - J(0)$$
$$\Delta 03I = I(3) - I(0)$$
$$\Delta 03J = J(3) - J(0)$$

P0	P1
P2	P3

$$P0 = C + I(0) * (A - C) + J(0) * (B - C)$$
$$P1 = P0 + \Delta 01I * (A - C) + \Delta 01J * (B - C)$$
$$P2 = P0 + \Delta 02I * (A - C) + \Delta 02J * (B - C)$$
$$P3 = P0 + \Delta 03I * (A - C) + \Delta 03J * (B - C)$$

P0 is computed at 20x24 mantissa precision and P1 to P3 are computed at 8X24 mantissa precision. So far no visual degradation of the image was seen using this scheme.

Multiples (Full Precision): 2
Multiplies (Reduced precision): 6
Subtracts 19x24 (Parameters): 2
Adds: 8

FORMAT OF P0’s IJ : Mantissa 20 Exp 4 for I + Sign
Mantissa 20 Exp 4 for J + Sign

FORMAT of Deltas (x3):Mantissa 8 Exp 4 for I + Sign
Mantissa 8 Exp 4 for J + Sign

Total number of bits : 20*2 + 8*6 + 4*8 + 4*2 = 128

All numbers are kept using the un-normalized floating point convention: if exponent is different than 0 the number is normalized if not, then the number is un-normalized. The maximum range for the IJs (Full precision) is +/- 63 and the range for the Deltas is +/- 127.

15.1 Interpolation of constant attributes

Because of the floating point imprecision, we need to take special provisions if all the interpolated terms are the same or if two of the barycentric coordinates are the same.

We start with the premise that if A = B and B = C and C = A, then P0,1,2,3 = A. Since one or more of the IJ terms may be zero, so we extend this to:

```
if (A=B and B=C and C=A)
  P0,1,2,3 = A;
else if ((I = 0) or (J = 0)) and
        ((J = 0) or (1-I-J = 0)) and
        ((1-J-I = 0) or (I = 0))) {
            if(I != 0) {
               P0 = A;
            } else if(J != 0) {
               P0 = B;
            } else {
               P0 = C;
           }
          //rest of the quad interpolated normally
}
else
{
          normal interpolation
}
```

16. -Staging Registers

Formatted: Bullets and Numbering

In order for the reuse of the vertices to be 14, the sequencer will have to re-order the data sent IN ORDER by the VGT for it to be aligned with the parameter cache memory arrangement. Given the following group of vertices sent by the VGT:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 || 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 || 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 || 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63

The sequencer will re-arrange them in this fashion:

0 1 2 3 16 17 18 19 32 33 34 35 48 49 50 51 || 4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55 || 8 9 10 11 24 25 26 27 40 41 42 43 56 57 58 59 || 12 13 14 15 28 29 30 31 44 45 46 47 60 61 62 63

The || markers show the SP divisions. In the event a shader pipe is broken, the VGT will send padding to account for the missing pipe. For example, if SP1 is broken, vertices 4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55 will still be sent by the VGT to the SQ **BUT** will not be processed by the SP and thus should be considered invalid (by the SU and VGT).

Formatted

The most straightforward, *non-compressed* interface method would be to convert, in the VGT, the data to 32-bit floating point prior to transmission to the VSISRs. In this scenario, the data would be transmitted to (and stored in) the VSISRs in full 32-bit floating point. This method requires three 24-bit fixed-to-float converters in the VGT. Unfortunately, it also requires and additional 3,072 bits of storage across the VSISRs. This interface is illustrated in Figure 2. The area of the fixed-to-float converters and the VSISRs for this method is roughly estimated as 0.759sqmm using the R300 process. The gate count estimate is shown in Figure 1.

Basis for 8-deep Latch Memory (from R300)			
8x24-bit	11631 μ^2		60.57813 μ^2 per bit
Area of 96x8-deep Latch Memory	46524 μ^2		
Area of 24-bit Fix-to-float Converter	4712 μ^2 per converter		
Method 1	Block	Quantity	Area
	F2F	3	14136
	8x96 Latch	16	744384
			758520 μ^2

Figure 1:Area Estimate for VGT to Shader Interface

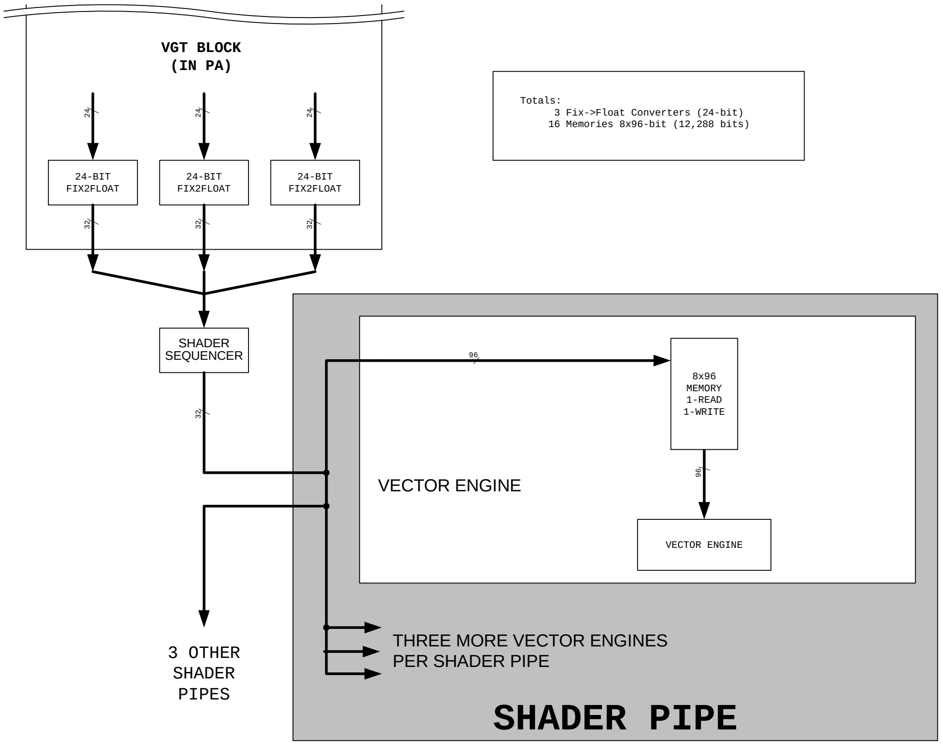


Figure 2:VGT to Shader Interface

~~16.~~17. The parameter cache

Formatted: Bullets and Numbering

The parameter cache is where the vertex shaders export their data. It consists of 16 128x128 memories (1R/1W). The reuse engine will make it so that all vertexes of a given primitive will hit different memories. The allocation method for these memories is a simple round robin. The parameter cache pointers are mapped in the following way: 4MSBs are the memory number and the 7 LSBs are the address within this memory.

MEMORY NUMBER 4 bits	ADDRESS 7 bits

The PA generates the parameter cache addresses as the positions comes from the SQ. All it needs to do is keep a Current_Location pointer (7 bits only) and as the positions comes increment the memory number. When the memory number field wraps around, the PA increments the Current_Location by VS_EXPORT_COUNT_6 (a snooped register from the SQ). As an example, say the memories are all empty to begin with and the vertex shader is exporting 8 parameters per vertex (VS_EXPORT_COUNT_6 = 8). The first position received is going to have the PC address 00000000000 the second one 00010000000, third one 00100000000 and so on up to 11110000000. Then the next position received (the 17th) is going to have the address 00000001000, the 18th 00010001000, the 19th 00100001000 and so on. The Current_location is NEVER reset BUT on chip resets. The only thing to be careful about is that if the SX doesn't send you a full group of positions (<64) then you need to fill the address space so that the next group starts correctly aligned (for example if you receive only 33 positions then you need to add 1*VS_EXPORT_COUNT_6 to Current_Location and reset the memory count to 0 before the next vector begins).

Formatted

Formatted

Formatted

~~17.~~18. Vertex position exporting

Formatted: Bullets and Numbering

On clause 3 the vertex shader can export to the PA both the vertex position and the point sprite. It can also do so at clause 7 if not done at clause 3. The storage needed to perform the position export is at least 64x128 memories for the position and 64x32 memories for the sprite size. It is going to be taken in the pixel output fifo from the SX blocks. The clause where the position export occurs is specified by the EXPORT_LATE register. If turned on, it means that the export is going to occur at ALU clause 7 if unset position export occurs at clause 3.

~~18.~~19. Exporting Arbitration

Formatted: Bullets and Numbering

Any type of exporting clause can be co-issued. The sequencer will have to make sure back to back memory exports (position/straight memory exports) are interleaved with NOPs as we don't have the bandwidth to service them at full speed.

~~Here are the rules for co-issuing exporting ALU clauses.~~

Formatted: Bullets and Numbering

~~1)Position exports and position exports cannot be co-issued.~~
~~2)Position exports and memory exports cannot be co-issued.~~
~~3)Position exports and Z/Color exports cannot be co-issued.~~

~~4)Memory exports and Z/Color exports cannot be co-issued.~~
~~5)Memory exports and memory exports cannot be co-issued.~~
~~6)Z/color exports and Z/color exports cannot be co-issued.~~
~~7)Parameter exports and Z/Color exports CAN be co-issued.~~

~~8)Parameter exports and parameter exports CAN be co-issued.~~
~~9)Parameter exports and memory exports CAN be co-issued.~~

~~19.~~20. Export Types

The export type (or the location where the data should be put) is specified using the destination address field in the ALU instruction. Here is a list of all possible export modes:

~~19.1~~20.1 Vertex Shading

Formatted: Bullets and Numbering

0:15	- 16 parameter cache
16:31	- Empty (Reserved?)
32:43	- 12 vertex exports to the frame buffer and index
44:47	- Empty
48:59	- 12 debug export (interpret as normal vertex export)
60	- export addressing mode
61	- Empty
62	- sprite size export that goes with position export (point_h,point_w,edgeflag,misc)
63	- position

~~19.2~~20.2 Pixel Shading

Formatted: Bullets and Numbering

0	- Color for buffer 0 (primary)
1	- Color for buffer 1
2	- Color for buffer 2
3	- Color for buffer 3
4:7	- Empty
8	- Buffer 0 Color/Fog (primary)
9	- Buffer 1 Color/Fog
10	- Buffer 2 Color/Fog
11	- Buffer 3 Color/Fog
12:15	- Empty
16:31	- Empty (Reserved?)
32:43	- 12 exports for multipass pixel shaders.
44:47	- Empty
48:59	- 12 debug exports (interpret as normal pixel export)
60	- export addressing mode
61:62	- Empty
63	- Z for primary buffer (Z exported to 'alpha' component)

~~20.~~21. Special Interpolation modes

Formatted: Bullets and Numbering

~~20.1~~21.1 Real time commands

We are unable to use the parameter memory since there is no way for a command stream to write into it. Instead we need to add three 16x128 memories (one for each of three vertices x 16 interpolants). These will be mapped onto the register bus and written by type 0 packets, and output to the the parameter busses (the sequencer and/or PA need to be able to address the realtime parameter memory as well as the regular parameter store. For higher performance we should be able able to view them as two banks of 16 and do double buffering allowing one to be loaded, while the other is rasterized with. Most overlay shaders will need 2 or 4 scalar coordinates, one option might be to restrict the memory to 16x64 or 32x64 allowing only two interpolated scalars per cycle, the only problem I see with this is, if we view support for 16 vector-4 interpolants important (true only if we map Microsoft's high priority stream to the realtime stream), then the PA/sequencer need to support a realtime-specific mode where we need to address 32 vectors of parameters instead of 16. This mode is triggered by the primitive type: REAL TIME.

~~20.2~~21.2 Sprites/ XY screen coordinates/ FB information

Formatted: Bullets and Numbering

When working with sprites, one may want to overwrite the parameter 0 with SC generated data. Also, XY screen coordinates may be needed in the shader program. This functionality is controlled by the gen_I0 register (in SQ) in conjunction with the SND_XY register (in SC). Also it is possible to send the faceness information (for OGL front/back special operations) to the shader using the same control register. Here is a list of all the modes and how they interact together:

Gen_st is a bit taken from the interface between the SC and the SQ. This is the MSB of the primitive type. If the bit is set, it means we are dealing with Point AA, Line AA or sprite and in this case the vertex values are going to generated between 0 and 1.

Param_Gen_I0 disable, snd_xy disable, no gen_st – I0 = No modification
Param_Gen_I0 disable, snd_xy disable, gen_st – I0 = No modification
Param_Gen_I0 disable, snd_xy enable, no gen_st – I0 = No modification
Param_Gen_I0 disable, snd_xy enable, gen_st – I0 = No modification
Param_Gen_I0 enable, snd_xy disable, no gen_st – I0 = garbage, garbage, garbage, faceness
Param_Gen_I0 enable, snd_xy disable, gen_st – I0 = garbage, garbage, s, t
Param_Gen_I0 enable, snd_xy enable, no gen_st – I0 = screen x, screen y, garbage, faceness
Param_Gen_I0 enable, snd_xy enable, gen_st – I0 = screen x, screen y, s, t

~~20.3~~21.3 Auto generated counters

Formatted: Bullets and Numbering

In the cases we are dealing with multipass shaders, the sequencer is going to generate a vector count to be able to both use this count to write the 1st pass data to memory and then use the count to retrieve the data on the 2nd pass. The count is always generated in the same way but it is passed to the shader in a slightly different way depending on the shader type (pixel or vertex). This is toggled on and off using the GEN_INDEX register. The sequencer is going to keep two counters, one for pixels and one for vertices. Every time a full vector of vertices or pixels is written to the GPRs the counter is incremented. Every time a state change is detected, the corresponding counter is reset. While there is only one count broadcast to the GPRs, the LSB are hardwired to specific values making the index different for all elements in the vector.

~~20.3.1~~21.3.1 *Vertex shaders*

Formatted: Bullets and Numbering

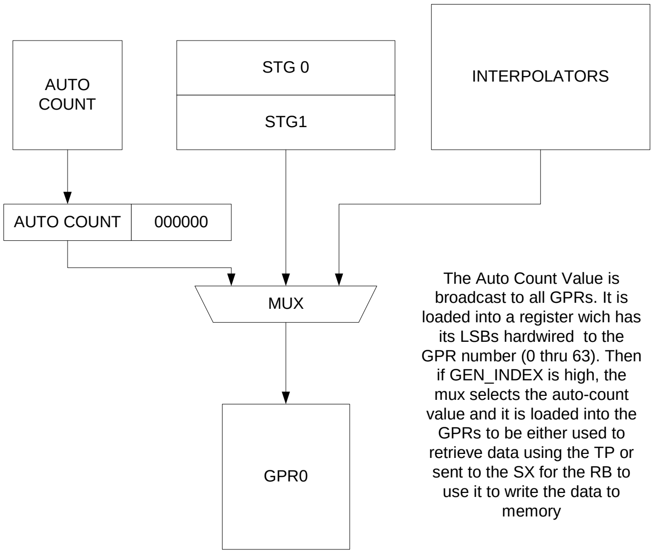
In the case of vertex shaders, if GEN_INDEX is set, the data will be put into the x field of the third register (it means that the compiler must allocate 3 GPRs in all multipass vertex shader modes).

~~20.3.2~~21.3.2 *Pixel shaders*

Formatted: Bullets and Numbering

In the case of pixel shaders, if GEN_INDEX is set and Param_Gen_I0 is enabled, the data will be put in the x field of the 2nd register (R~~l~~1.x), else if GEN_INDEX is set the data will be put into the x field of the 1st register (R0.x).

Formatted



The Auto Count Value is broadcast to all GPRs. It is loaded into a register wich has its LSBs hardwired to the GPR number (0 thru 63). Then if GEN_INDEX is high, the mux selects the auto-count value and it is loaded into the GPRs to be either used to retrieve data using the TP or sent to the SX for the RB to use it to write the data to memory

~~21.~~22. State management

Formatted: Bullets and Numbering

Every clock, the sequencer will report to the CP the oldest states still in the pipe. These are the states of the programs as they enter the last ALU clause.

~~21.1~~22.1 Parameter cache synchronization

Formatted: Bullets and Numbering

In order for the sequencer not to begin a group of pixels before the associated group of vertices has finished, the sequencer will keep a 6 bit count per state (for a total of 8 counters). These counters are initialized to 0 and every time a vertex shader exports its data TO THE PARAMETER CACHE, the corresponding pointer is incremented. When the SC sends a new vector of pixels with the SC_SQ_new_vector bit asserted, the sequencer will first check if the count is greater than 0 before accepting the transmission (it will in fact accept the transmission but then lower its ready to receive). Then the sequencer waits for the count to go to one and decrements it. The sequencer can then issue the group of pixels to the interpolators. Every time the state changes, the new state counter is initialized to 0.

~~22.~~23. XY Address imports

Formatted: Bullets and Numbering

The SC will be able to send the XY addresses to the GPRs. It does so by interleaving the writes of the IJs (to the IJ buffer) with XY writes (to the XY buffer). Then when writing the data to the GPRs, the sequencer is going to interpolate the IJ data or pass the XY data thru a Fix→float converter and expander and write the converted values to the GPRs. The Xys are currently SCREEN SPACE COORDINATES. The values in the XY buffers will wrap. See section 21.2~~21.2~~~~20.2~~ for details on how to control the interpolation in this mode.

~~22.1~~23.1 Vertex indexes imports

Formatted: Bullets and Numbering

In order to import vertex indexes, we have ~~64x2x96~~ 16 8x96 staging registers. These are loaded one line at a time by the VGT block (96 bits). They are loaded in floating point format and can be transferred in 4 or 8 clocks to the GPRs.

~~23.~~24. Registers

Formatted: Bullets and Numbering

~~23.1~~24.1 Control

REG_DYNAMIC	Dynamic allocation (pixel/vertex) of the register file on or off.
REG_SIZE_PIX	Size of the register file's pixel portion (minimal size when dynamic allocation turned on)
REG_SIZE_VTX	Size of the register file's vertex portion (minimal size when dynamic allocation turned on)
ARBITRATION_POLICY	policy of the arbitration between vertexes and pixels
INST_STORE_ALLOC	interleaved, separate
INST_BASE_VTX	start point for the vertex instruction store (RT always ends at vertex_base and Begins at 0)
INST_BASE_PIX	start point for the pixel shader instruction store
ONE_THREAD	debug state register. Only allows one program at a time into the GPRs
ONE_ALU	debug state register. Only allows one ALU program at a time to be executed (instead of 2)
INSTRUCTION_ADDR	This is where the CP puts the base address of the instruction writes and type (auto-incremented on reads/writes)
INSTRUCTION_DATA	This is where the CP puts the actual data going to the instruction memory
~~CONSTANT_DATA~~	~~This is where the CP puts constant data (32 bits)~~
~~CONSTANT_ADDR~~	~~This is where the CP puts the logical constant address (9 bits)~~CONSTANTS 512*4 ALU constants + 32*6 Texture state 32 bits registers (logically mapped)
INSTRUCTION_ADDR_RT	This is where the CP puts the base address of the instruction writes and type for Real Time (auto-incremented on reads/writes)
INSTRUCTION_DATA_RT	This is where the CP puts the actual data going to the instruction memory for Real Time
~~CONSTANT_DATA_RT~~	~~This is where the CP puts constant data for Real Time (32 bits)~~

~~CONSTANT_ADDR_RT~~	~~This is where the CP puts the logical constant address for Real Time (9 bits)~~CONSTANTS_RT 256*4 ALU constants + 32*6 texture states? (physically mapped)
CONSTANT_EO_RT	This is the size of the space reserved for real time in the constant store (from 0 to CONSTANT_EO_RT). The re-mapping table operates on the rest of the memory
EXPORT_LATE	Controls whether or not we are exporting position from clause 3. If set, position exports occur at clause 7.

~~23.2~~24.2 Context

Formatted: Bullets and Numbering

VS_FETCH_{0…7}	eight 8 bit pointers to the location where each clauses control program is located
VS_ALU_{0…7}	eight 8 bit pointers to the location where each clauses control program is located
PS_FETCH_{0…7}	eight 8 bit pointers to the location where each clauses control program is located
PS_ALU_{0…7}	eight 8 bit pointers to the location where each clauses control program is located
PS_BASE	base pointer for the pixel shader in the instruction store
VS_BASE	base pointer for the vertex shader in the instruction store
VS_CF_SIZE	size of the vertex shader (# of instructions in control program/2)
PS_CF_SIZE	size of the pixel shader (# of instructions in control program/2)
PS_SIZE	size of the pixel shader (cntl+instructions)
VS_SIZE	size of the vertex shader (cntl+instructions)
PS_NUM_REG	number of GPRs to allocate for pixel shader programs
VS_NUM_REG	number of GPRs to allocate for vertex shader programs
PARAM_SHADE	One 16 bit register specifying which parameters are to be gouraud shaded (0 = flat, 1 = gouraud)
PARAM_WRAP	64 bits: for which parameters (and channels (xyzw)) do we do the cyl wrapping (0=linear, 1=cylindrical).
PS_EXPORT_MODE	0xxxx : Normal mode 1xxxx : Multipass mode If normal, bbbz where bbb is how many colors (0-4) and z is export z or not If multipass 1-12 exports for color.
VS_EXPORT_MASK	which of the last 6 ALU clauses is exporting (multipass only)
VS_EXPORT_MODE	0: position (1 vector), 1: position (2 vectors), 3:multipass
VS_EXPORT _COUNT_{0…6}	Six 4 bit counters representing the # of interpolated parameters exported in clause 7 (located in VS_EXPORT_COUNT_6) OR # of exported vectors to memory per clause in multipass mode (per clause)
PARAM_GEN_I0	Do we overwrite or not the parameter 0 with XY data and generated T and S values
GEN_INDEX	Auto generates an address from 0 to XX. Puts the results into ~~R1~~ R0-1 for pixel shaders and ~~R3~~ R2 for vertex shaders
CONST_BASE_VTX (9 bits)	Logical Base address for the constants of the Vertex shader
CONST_BASE_PIX (9 bits)	Logical Base address for the constants of the Pixel shader
CONST_SIZE_PIX (8 bits)	Size of the logical constant store for pixel shaders
CONST_SIZE_VTX (8 bits)	Size of the logical constant store for vertex shaders
INST_PRED_OPTIMIZE	Turns on the predicate bit optimization (if of, conditional_execute_predicates is always executed).
CF_BOOLEANS	256 boolean bits
CF_LOOP_COUNT	32x8 bit counters (number of times we traverse the loop)
CF_LOOP_START	32x8 bit counters (init value used in index computation)
CF_LOOP_STEP	32x8 bit counters (step value used in index computation)

~~24.~~25. DEBUG Registers

Formatted: Bullets and Numbering

~~24.1~~25.1 Context

DB_PROB_ADDR	instruction address where the first problem occurred
DB_PROB_COUNT	number of problems encountered during the execution of the program
DB_INST_COUNT	instruction counter for debug method 2
DB_BREAK_ADDR	break address for method number 2

DB_CLAUSE
_MODE_ALU_{0…7}   clause mode for debug method 2 (0: normal, 1: addr, 2: kill)
DB_CLAUSE
_MODE_FETCH_{0…7}   clause mode for debug method 2 (0: normal, 1: addr, 2: kill)

Formatted: Bullets and Numbering

~~25.~~26. Interfaces

~~25.1~~26.1 External Interfaces

Whenever an x is used, it means that the bus is broadcast to all units of the same name. For example, if a bus is named SQ→SPx it means that SQ is going to broadcast the same information to all SP instances.

Formatted: Bullets and Numbering

~~25.1.1~~26.1.1 *SC to SQ : IJ Control bus*

This is the control information sent to the sequencer in order to control the IJ fifos and all other information needed to execute a shader program on the sent pixels. This information is sent over 2 clocks, if SENDXY is asserted the next control packet is going to be ignored and XY information is going to be sent on the IJ bus (for the quads that where just sent). All pixels from the group of quads are from the same primitive, all quads of a vector are from the same render state.

Name	Direction	Bits	Description
SC_SQ_q_wr_mask	SC→SQ	4	Quad Write mask left to right
SC_SQ_lod_correct	SC→SQ	24	LOD correction per quad (6 bits per quad)
SC_SQ_flat_vertex	SC→SQ	2	Provoking vertex for flat shading
SC_SQ_param_ptr0	SC→SQ	11	P Store pointer for vertex 0
SC_SQ_param_ptr1	SC→SQ	11	P Store pointer for vertex 1
SC_SQ_param_ptr2	SC→SQ	11	P Store pointer for vertex 2
SC_SQ_end_of_vect	SC→SQ	1	End of the vector
SC_SQ_store_dealloc	SC→SQ	1	Deallocation token for the P Store
SC_SQ_state	SC→SQ	3	State/constant pointer
SC_SQ_valid_pixel	SC→SQ	16	Valid bits for all pixels
SC_SQ_null_prim	SC→SQ	1	Null Primitive (for PC deallocation purposes)
SC_SQ_end_of_prim	SC→SQ	1	End Of the primitive
SC_SQ_send_xy	SC→SQ	1	Sending XY information [XY information is going to be sent on the next clock]
SC_SQ_prim_type	SC→SQ	3	Real time command need to load tex cords from alternate buffer. Line AA, Point AA and Sprite reads their parameters from GEN_T and GEN_S GPRs. 000 : Normal 011 : Real Time 100 : Line AA 101 : Point AA 110 : Sprite
SC_SQ_new_vector	SC→SQ	1	This primitive comes from a new vector of vertices. Make sure that the corresponding vertex shader has finished before starting the group of pixels.
SC_SQ_RTRn	SQ→SC	1	Stalls the PA in n clocks
SC_SQ_RTS	SC→SQ	1	SC ready to send data

Formatted: Bullets and Numbering

~~25.1.2~~26.1.2 *SQ to SP: Interpolator bus*

Name	Direction	Bits	Description
SQ_SPx_interp_prim_type	SQ→SPx	3	Type of the primitive 000 : Normal 011 : Real Time 100 : Line AA 101 : Point AA 110 : Sprite

SQ_SPx_interp_flat_vtx	SQ→SPx	2	Provoking vertex for flat shading
SQ_SPx_interp_flat_gouraud	SQ→SPx	1	Flat or gouraud shading
SQ_SPx_interp_cyl_wrap	SQ→SPx	4	Wich parameter needs to be cylindrical wrapped
SQ_SPx_interp_ijline	SQ→SPx	2	Line in the IJ/XY buffer to use to interpolate
SQ_SPx_interp_buff_swap	SQ→SPx	1	Swap the IJ/XY buffers at the end of the interpolation
SQ_SPx_interp_gen_I0	SQ→SPx	1	Generate I0 or not. This tells the interpolators not to use the parameter cache but rather overwrite the data with interpolated 1 and 0. Overwrite if gen_I0 is high.

~~25.1.3SQ to SP: GPR Input Mux select~~

Formatted: Bullets and Numbering

~~This interface is synchronized with the Interpolator bus. This controls the input mux to the GPRs. The three types of data are: generated index, Interpolated data, vertex index data (coming from the staging registers).~~

~~25.1.4~~26.1.3 SQ to SP: Parameter Cache Read control bus

Formatted: Bullets and Numbering

The four following interfaces (SQ→SP, SQ→SX,SP→SX and SX→Interpolators) are all SYNCHRONIZED together.

Name	Direction	Bits	Description
SQ_SPx_ptr0	SQ→SPx	9	Pointer of PC
SQ_SPx_ptr1	SQ→SPx	9	Pointer of PC
SQ_SPx_ptr2	SQ→SPx	9	Pointer of PC
SQ_SP0_read_ena	SQ→SP0	4	Read enables for the 4 memories in the SP0
SQ_SP1_read_ena	SQ→SP1	4	Read enables for the 4 memories in the SP1
SQ_SP2_read_ena	SQ→SP2	4	Read enables for the 4 memories in the SP2
SQ_SP3_read_ena	SQ→SP3	4	Read enables for the 4 memories in the SP3

~~25.1.5~~26.1.4 SQ to SX: Parameter Cache Mux control Bus

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SXx_mux0	SQ→SXx	4	Mux control for PC (4 MSbs of Pointer)
SQ_SXx_mux1	SQ→SXx	4	Mux control for PC (4 MSbs of Pointer)
SQ_SXx_mux2	SQ→SXx	4	Mux control for PC (4 MSbs of Pointer)

~~25.1.6~~26.1.5 SQ to SP: Staging Register Data

Formatted: Bullets and Numbering

This is a broadcast bus that sends the VSISR information to the staging registers of the shader pipes.

Name	Direction	Bits	Description
SQ_SPx~~0~~_vgt_vsisr_data	SQ→SPx~~0~~	96	Pointers of indexes or HOS surface information
SQ_SP~~0~~x_vgt_vsisr_double	SQ→SPx~~0~~	1	0: Normal 96 bits per vert 1: double 192 bits per vert
SQ_SP0_data_valid	SQ→SP0	1	Data is valid
SQ_SP1_data_valid	SQ→SP1	1	Data is valid
SQ_SP2_data_valid	SQ→SP2	1	Data is valid
SQ_SP3_data_valid	SQ→SP3	1	Data is valid

~~25.1.7~~26.1.6 PA to SQ : Vertex interface

Formatted: Bullets and Numbering

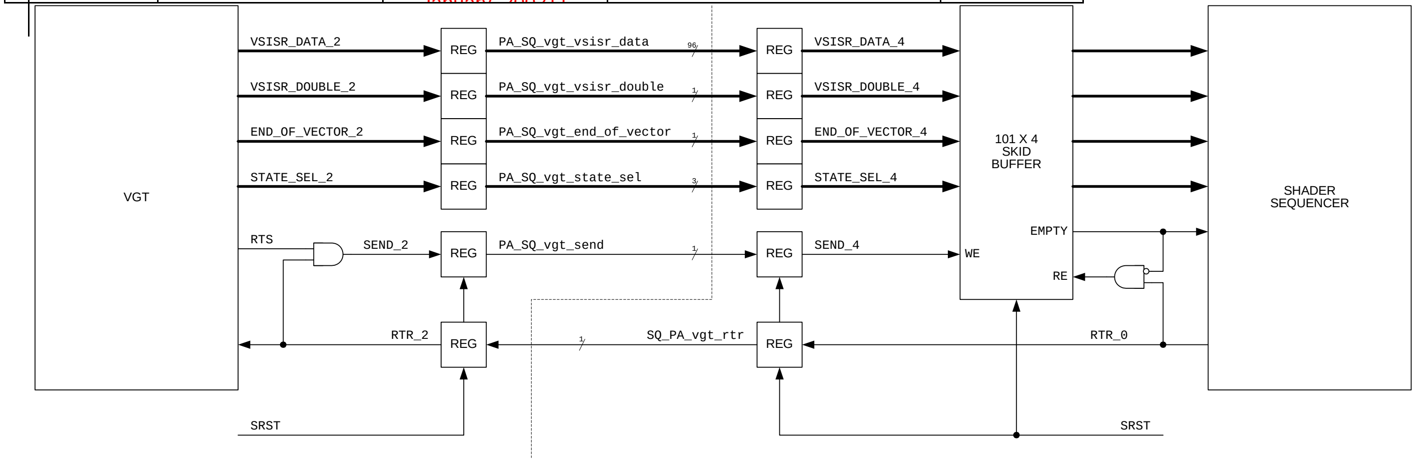
~~25.1.7.1~~26.1.6.1 Interface Signal Table

The area difference between the two methods is not sufficient to warrant complicating the interface or the state requirements of the VSISRs. **Therefore, the POR for this interface is that the VGT will transmit the data to the VSISRs (via the Shader Sequencer) in full, 32-bit floating-point format.** The VGT can transmit up to six 32-bit floating-point values to each VSISR where four or more values require two transmission clocks. The data bus is 96 bits wide.

Name	Bits	Description
PA_SQ_vgt_vsisr_data	96	Pointers of indexes or HOS surface information
PA_SQ_vgt_vsisr_double	1	0: Normal 96 bits per vert 1: double 192 bits per vert
PA_SQ_vgt_end_of_vector	1	Indicates the last VSISR data set for the current process vector (for double vector data, "end_of_vector" is set on the second vector)
PA_SQ_vgt_state	3	Render State (6*3+3 for constants). This signal is guaranteed to be correct when “PA_SQ_vgt_end_of_vector” is high.
PA_SQ_vgt_send	1	Data on the VGT_SQ is valid receive (see write-up for standard R400 SEND/RTR interface handshaking)
SQ_PA_vgt_rtr	1	Ready to receive (see write-up for standard R400 SEND/RTR interface handshaking)

~~25.1.7.2~~26.1.6.2 Interface Diagrams

Formatted: Bullets and Numbering


VGT
VSISR_DATA_2
VSISR_DOUBLE_2
END_OF_VECTOR_2
STATE_SEL_2
RTS
SEND_2
RTR_2
SRST
REG
PA_SQ_vgt_vsisr_data
96
PA_SQ_vgt_vsisr_double
1
PA_SQ_vgt_end_of_vector
1
PA_SQ_vgt_state_sel
3
PA_SQ_vgt_send
1
SQ_PA_vgt_rtr
1
VSISR_DATA_4
VSISR_DOUBLE_4
END_OF_VECTOR_4
STATE_SEL_4
SEND_4
101 X 4
SKID
BUFFER
EMPTY
WE
RE
RTR_0
SRST
SHADER
SEQUENCER

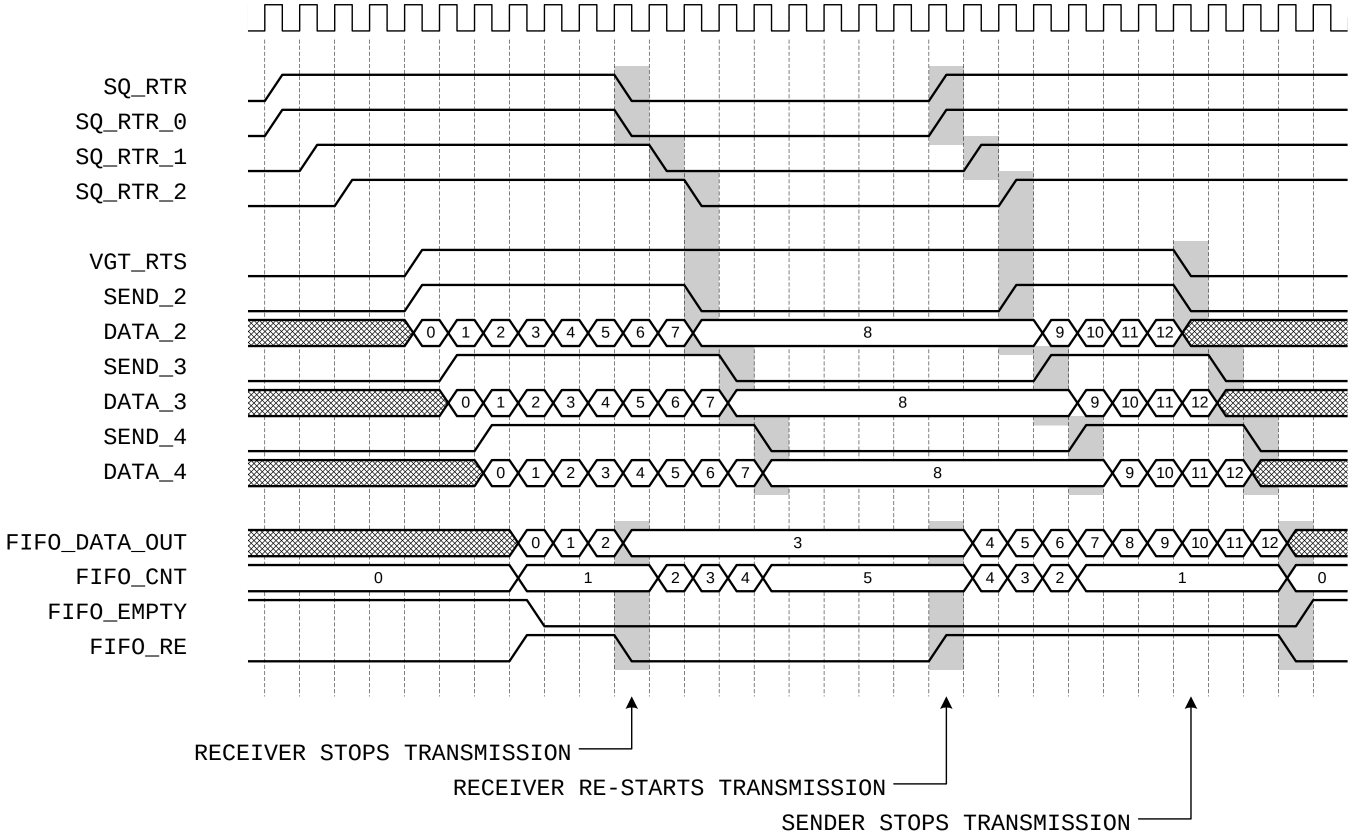


Figure 1. Detailed Logical Diagram for PA_SQ_vgt Interface.

~~25.1.8~~26.1.7 SQ to CP: State report

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_CP_vrtx_ state	SEQ→CP	3	Oldest vertex state still in the pipe
SQ_CP_pix_state	SEQ→CP	3	Oldest pixel state still in the pipe

~~25.1.9~~26.1.8 SQ to SX: Control bus

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SXx_exp_Pixel	SQ→SXx	1	1: Pixel 0: Vertex
SQ_SXx_exp_start	SQ→SXx	1	Raised to indicate that the SQ is starting an export
SQ_SXx_exp_Clause	SQ→SXx	3	Clause number, which is needed for vertex clauses
SQ_SXx_exp_State	SQ→SXx	3	State ID, which is needed for vertex clauses

These fields are sent synchronously with SP export data, described in SP0→SX0 interface
{ISSUE: Where are the PC pointers}

~~25.1.10~~26.1.9 SX to SQ : Output file control

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SXx_SQ_Export_count_rdy	SXx→SQ	1	Raised by SX0 to indicate that the following two fields reflect the result of the most recent export
SXx_SQ_Export_Position	SXx→SQ	1	Specifies whether there is room for another position.
SXx_SQ_Export_Buffer	SXx→SQ	7	Specifies the space available in the output buffers. 0: buffers are full 1: 2K-bits available (32-bits for each of the 64 pixels in a clause) ... 64: 128K-bits available (16 128-bit entries for each of 64 pixels) 65-127: RESERVED

~~25.1.11~~26.1.10 SQ to TP: Control bus

Formatted: Bullets and Numbering

Once every clock, the fetch unit sends to the sequencer on which clause it is now working and if the data in the GPRs is ready or not. This way the sequencer can update the fetch counters for the reservation station fifos. The sequencer also provides the instruction and constants for the fetch to execute and the address in the register file where to write the fetch return data.

Name	Direction	Bits	Description
TPx_SQ_data_rdy	TPx→ SQ	1	Data ready
TPx_SQ_clause_num	TPx→ SQ	3	Clause number
SQ_TPx_const	SQ→TPx	~~64~~48	Fetch state sent over 4 clocks (192 bits total)
SQ_TPx_instuct	SQ→TPx	24	Fetch instruction sent over 4 clocks
SQ_TPx_end_of_clause	SQ→TPx	1	Last instruction of the clause
SQ_TPx_phase	SQ→TPx	2	Write phase signal
SQ_TP0_lod_correct	SQ→TP0	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP0_pmask	SQ→TP0	4	Pixel mask 1 bit per pixel
SQ_TP1_lod_correct	SQ→TP1	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP1_pmask	SQ→TP1	4	Pixel mask 1 bit per pixel
SQ_TP2_lod_correct	SQ→TP2	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP2_pmask	SQ→TP2	4	Pixel mask 1 bit per pixel
SQ_TP3_lod_correct	SQ→TP3	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP3_pmask	SQ→TP3	4	Pixel mask 1 bit per pixel
SQ_TPx_clause_num	SQ→TPx	3	Clause number
SQ_TPx_write_gpr_index	SQ->TPx	7	Index into Register file for write of returned Fetch Data

~~25.1.12~~26.1.11 TP to SQ: Texture stall

Formatted: Bullets and Numbering

The TP sends this signal to the SQ when its input buffer is full. The SQ is going to send it to the SP X clocks after reception (maximum of 3 clocks of pipeline delay).

Name	Direction	Bits	Description
TP_SQ_fetch_stall	TP→ SQ	1	Do not send more texture request if asserted

~~25.1.13~~26.1.12 SQ to SP: Texture stall

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SPx_fetch_stall	SQ→SPx	1	Do not send more texture request if asserted

~~25.1.14~~26.1.13 SQ to SP: GPR, ~~and~~ Parameter cache control and auto counter

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SPx_wr_addr	SQ→SPx	7	Write address
SQ_SPx_gpr_rd_addr	SQ→SPx	7	Read address
SQ_SPx_gpr_re_addr	SQ→SPx	1	Read Enable
SQ_SPx_gpr_we_addr	SQ→SPx	1	Write Enable for the GPRs
SQ_SPx_gpr_phase_mux	SQ→SPx	2	The phase mux (arbitrates between inputs, ALU SRC reads and writes)
SQ_SPx_channel_mask	SQ→SPx	4	The channel mask
SQ_SP0_pixel_mask	SQ→SP0	4	The pixel mask
SQ_SP1_pixel_mask	SQ→SP1	4	The pixel mask
SQ_SP2_pixel_mask	SQ→SP2	4	The pixel mask
SQ_SP3_pixel_mask	SQ→SP3	4	The pixel mask
SQ_SPx_pc_we_addr	SQ→SPx	1	Write Enable for the parameter caches
SQ_SPx_gpr_input_mux	SQ→SPx	2	When the phase mux selects the inputs this tells from which source to read from: Interpolated data, VTX0, VTX1, autogen counter.
SQ_SPx_index_count	SQ→SPx	12?	Index count, common for all shader pipes

~~25.1.15~~26.1.14 SQ to SPx: Instructions

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SPx_instruct_start	SQ→SPx	1	Instruction start
SQ_SP_instruct	SQ→SPx	20	Instruction sent over 4 clocks
SQ_SPx_stall	SQ→SPx	1	Stall signal
SQ_SPx_export_count	SQ→SPx	3	Each set of four pixels or vectors is exported over eight clocks. This field specifies where the SP is in that sequence.
SQ_SPx_export_last	SQ→SPx	1	Asserted on the first shader count of the last export of the clause
SQ_SP0_export_pvalid	SQ→SP0	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP0_export_wvalid	SQ→SP0	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors
SQ_SP1_ export_pvalid	SQ→SP1	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP1_ export_wvalid	SQ→SP1	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors
SQ_SP2_ export_pvalid	SQ→SP2	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP2_ export_wvalid	SQ→SP2	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors
SQ_SP3_ export_pvalid	SQ→SP3	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP3_ export_wvalid	SQ→SP3	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors

~~25.1.16~~26.1.15 SP to SQ: Constant address load

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SP0_SQ_const_addr	SP0→SQ	36	Constant address load to the sequencer
SP0_SQ_valid	SP0→SQ	1	Data valid
SP1_SQ_const_addr	SP1→SQ	36	Constant address load to the sequencer
SP1_SQ_valid	SP1→SQ	1	Data valid
SP2_SQ_const_addr	SP2→SQ	36	Constant address load to the sequencer
SP2_SQ_valid	SP2→SQ	1	Data valid
SP3_SQ_const_addr	SP3→SQ	36	Constant address load to the sequencer
SP3_SQ_valid	SP3→SQ	1	Data valid

~~25.1.17~~26.1.16 SQ to SPx: constant broadcast

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SPx_constant	SQ→SPx	128	Constant broadcast

~~25.1.18~~26.1.17 SP0 to SQ: Kill vector load

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SP0_SQ_kill_vect	SP0→SQ	4	Kill vector load
SP1_SQ_kill_vect	SP1→SQ	4	Kill vector load
SP2_SQ_kill_vect	SP2→SQ	4	Kill vector load
SP3_SQ_kill_vect	SP3→SQ	4	Kill vector load

~~25.1.19~~26.1.18 SQ to CP: RBBM bus

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_RBB_rs	SQ→CP	1	Read Strobe
SQ_RBB_rd	SQ→CP	32	Read Data
SQ_RBBM_nrtrtr	SQ→CP	1	Optional
SQ_RBBM_rtr	SQ→CP	1	Real-Time (Optional)

~~25.1.20~~26.1.19 CP to SQ: RBBM bus

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
rbbm_we	CP→SQ	1	Write Enable
rbbm_a	CP→SQ	18	Address -- Upper Extent is TBD
rbbm_wd	CP→SQ	32	Data
rbbm_be	CP→SQ	4	Byte Enables
rbbm_re	CP→SQ	1	Read Enable
rbb_rs0	CP→SQ	1	Read Return Strobe 0
rbb_rs1	CP→SQ	1	Read Return Strobe 1
rbb_rd0	CP→SQ	32	Read Data 0
rbb_rd1	CP→SQ	32	Read Data 0
RBBM_SQ_soft_reset	CP→SQ	1	Soft Reset

~~26.~~27. Examples of program executions

Formatted: Bullets and Numbering

~~26.1.1~~27.1.1 Sequencer Control of a Vector of Vertices

1. PA sends a vector of 64 vertices (actually vertex indices – 32 bits/index for 2048 bit total) to the RE's Vertex FIFO
   - state pointer as well as tag into position cache is sent along with vertices
   - space was allocated in the position cache for transformed position before the vector was sent
   - **also before the vector is sent to the RE, the CP has loaded the global instruction store with the vertex shader program (using the MH?)**
   - The vertex program is assumed to be loaded when we receive the vertex vector.
     - the SEQ then accesses the IS base for this shader using the local state pointer (provided to all sequencers by the RBBM when the CP is done loading the program)

2. SEQ arbitrates between the Pixel FIFO and the Vertex FIFO – basically the Vertex FIFO always has priority
   - at this point the vector is removed from the Vertex FIFO
   - the arbiter is not going to select a vector to be transformed if the parameter cache is full unless the pipe as nothing else to do (ie no pixels are in the pixel fifo).

3. SEQ allocates space in the SP register file for index data plus GPRs used by the program
   - the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer that came down with the vertices
   - SEQ will not send vertex data until space in the register file has been allocated

4. SEQ sends the vector to the SP register file over the RE_SP interface (which has a bandwidth of 2048 bits/cycle)
   - the 64 vertex indices are sent to the 64 register files over 4 cycles
     - RF0 of SU0, SU1, SU2, and SU3 is written the first cycle
     - RF1 of SU0, SU1, SU2, and SU3 is written the second cycle
     - RF2 of SU0, SU1, SU2, and SU3 is written the third cycle
     - RF3 of SU0, SU1, SU2, and SU3 is written the fourth cycle

- the index is written to the least significant 32 bits **(floating point format?) (what about compound indices)** of the 128-bit location within the register file (w); the remaining data bits are set to zero (x, y, z)

5. SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of fetch state machine 0, or TSM0 FIFO)
    - the control packet contains the state pointer, the tag to the position cache and a register file base pointer.
6. TSM0 accepts the control packet and fetches the instructions for fetch clause 0 from the global instruction store
    - TSM0 was first selected by the TSM arbiter before it could start
7. all instructions of fetch clause 0 are issued by TSM0
8. the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)
    - TSM0 does not wait for requests made to the Fetch Unit to complete; it passes the register file write index for the fetch data to the TU, which will write the data to the RF as it is received
    - once the TU has written all the data to the register files, it increments a counter that is associated with ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go ahead start to execute the ALU clause
9. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store
10. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of fetch state machine 1, or TSM1 FIFO)
11. the control packet continues to travel down the path of reservation stations until all clauses have been executed
    - position can be exported in ALU clause 3 (or 4?); the data (and the tag) is sent over a position bus (which is shared with all four shader pipes) back to the PA's position cache
    - A parameter cache pointer is also sent along with the position data. This tells to the PA where the data is going to be in the parameter cache.
        - there is a position export FIFO in the SP that buffers position data before it gets sent back to the PA
        - the ASM arbiter will prevent a packet from starting an exporting clause if the position export FIFO is full
    - parameter data is exported in clause 7 (as well as position data if it was not exported earlier)
        - parameter data is sent to the Parameter Cache over a dedicated bus
        - the SEQ allocates storage in the Parameter Cache, and the SEQ deallocates that space when there is no longer a need for the parameters (it is told by the PA when using a token).
        - the ASM arbiter will prevent a packet from starting on ASM7 if the parameter cache (or the position buffer if position is being exported) is full
12. after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program

~~26.1.2~~27.1.2 *Sequencer Control of a Vector of Pixels*

Formatted: Bullets and Numbering

1. **As with vertex shader programs, pixel shaders are loaded into the global instruction store by the CP**
    - At this point it is assumed that the pixel program is loaded into the instruction store and thus ready to be read.
2. the RE's Pixel FIFO is loaded with the barycentric coordinates for pixel quads by the detailed walker
    - the state pointer and the LOD correction bits are also placed in the Pixel FIF0
    - the Pixel FIFO is wide enough to source four quad's worth of barycentrics per cycle
3. SEQ arbitrates between Pixel FIFO and Vertex FIFO – when there are no vertices pending OR there is no space left in the register files for vertices, the Pixel FIFO is selected
4. SEQ allocates space in the SP register file for all the GPRs used by the program
    - the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer
    - SEQ will not allow interpolated data to be sent to the shader until space in the register file has been allocated
5. SEQ controls the transfer of interpolated data to the SP register file over the RE_SP interface (which has a bandwidth of 2048 bits/cycle). See interpolated data bus diagrams for details.

6. SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of fetch state machine 0, or TSM0 FIFO)
    - note that there is a separate set of reservation stations/arbiters/state machines for vertices and for pixels
    - the control packet contains the state pointer, the register file base pointer, and the LOD correction bits
    - all other information (such as quad address for example) travels in a separate FIFO
7. TSM0 accepts the control packet and fetches the instructions for fetch clause 0 from the global instruction store
    - TSM0 was first selected by the TSM arbiter before it could start
8. all instructions of fetch clause 0 are issued by TSM0
9. the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)
    - TSM0 does not wait for fetch requests made to the Fetch Unit to complete; it passes the register file write index for the fetch data to the TU, which will write the data to the RF as it is received
    - once the TU has written all the data for a particular clause to the register files, it increments a counter that is associated with the ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go ahead and pop the FIFO and start to execute the ALU clause
10. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store
11. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of fetch state machine 1, or TSM1 FIFO)
12. the control packet continues to travel down the path of reservation stations until all clauses have been executed
    - pixel data is exported in the last ALU clause (clause 7)
        - it is sent to an output FIFO where it will be picked up by the render backend
        - the ASM arbiter will prevent a packet from starting on ASM7 if the output FIFO is full
13. after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program

~~26.1.3~~27.1.3 *Notes*

Formatted: Bullets and Numbering

14. The state machines and arbiters will operate ahead of time so that they will be able to immediately start the real threads or stall.
15. The register file base pointer for a vector needs to travel with the vector through the reservation stations, but the instruction store base pointer does not – this is because the RF pointer is different for all threads, but the IS pointer is only different for each state and thus can be accessed via the state pointer.

~~27.~~28. Open issues

Formatted: Bullets and Numbering

Need to do some testing on the size of the register file as well as on the register file allocation method (dynamic VS static).

Saving power?

	ORIGINATE DATE 24 September, 2001	EDIT DATE 4 September, 20154 February, 20029	DOCUMENT-REV. NUM. GEN-CXXXXX-REVA	PAGE 1 of 50

Author: Laurent Lefebvre

Issue To:	**Copy No:**

R400 Sequencer Specification

SQ

Version 1.7~~6~~

Overview: This is an architectural specification for the R400 Sequencer block (SEQ). It provides an overview of the required capabilities and expected uses of the block. It also describes the block interfaces, internal sub-blocks, and provides internal state diagrams.

AUTOMATICALLY UPDATED FIELDS:
Document Location: C:\perforce\r400\doc_lib\design\blocks\sq\R400_Sequencer.doc
Current Intranet Search Title: R400 Sequencer Specification

APPROVALS	
Name/Dept	Signature/Date

Remarks:

THIS DOCUMENT CONTA~~read_ptr~~INS CONFIDENTIAL INFORMATION THAT COULD BE SUBSTANTIALLY DETRIMENTA~~read_ptr~~L TO THE INTEREST OF ATI TECHNOLOGIES INC. THROUGH UNAUTHORIZED USE OR DISCLOSURE.

Table Of Contents

1. OVERVIEW .... **7~~6~~**
1.1 Top Level Block Diagram .... 9~~8~~
1.2 Data Flow graph (SP) .... 11~~10~~
1.3 Control Graph .... 12~~11~~
2. INTERPOLATED DATA BUS .... **12~~11~~**
3. INSTRUCTION STORE .... **15~~14~~**
4. SEQUENCER INSTRUCTIONS .... **17~~16~~**
5. CONSTANT STORES .... **17~~16~~**
5.1 Memory organizations .... 17~~16~~
5.2 Management of the re-mapping tables .... 17~~16~~
5.2.1 Dirty bits .... 20~~18~~
5.2.2 Free List Block .... 20~~18~~
5.2.3 De-allocate Block .... 21~~18~~
5.2.4 Operation of Incremental model .... 21~~18~~
5.3 Constant Store Indexing .... 21~~19~~
5.4 Real Time Commands .... 22~~19~~
5.5 Constant Waterfalling .... 22~~19~~
6. LOOPING AND BRANCHES .... **23~~20~~**
6.1 The controlling state. .... 23~~20~~
6.2 The Control Flow Program .... 23~~20~~
6.3 Data dependant predicate instructions .... 25~~23~~
6.4 HW Detection of PV,PS .... 26~~23~~
6.5 Register file indexing .... 26~~23~~
6.6 Predicated Instruction support for Texture clauses .... 27~~24~~
6.7 Debugging the Shaders .... 27~~24~~
6.7.1 Method 1: Debugging registers .... 27~~24~~
6.7.2 Method 2: Exporting the values in the GPRs (12) .... 27~~25~~
7. PIXEL KILL MASK .... **28~~25~~**
8. MULTIPASS VERTEX SHADERS (HOS) .... **28~~25~~**
9. REGISTER FILE ALLOCATION .... **28~~25~~**
10. FETCH ARBITRATION .... **29~~26~~**
11. ALU ARBITRATION .... **29~~26~~**
12. HANDLING STALLS .... **30~~27~~**
13. CONTENT OF THE RESERVATION STATION FIFOS .... **30~~27~~**
14. THE OUTPUT FILE .... **30~~27~~**
15. IJ FORMAT .... **30~~27~~**
15.1 Interpolation of constant attributes .... 31~~28~~
16. STAGING REGISTERS .... **31~~28~~**
17. THE PARAMETER CACHE .... **33~~30~~**
18. VERTEX POSITION EXPORTING .... **34~~30~~**
19. EXPORTING ARBITRATION .... **34~~30~~**
20. EXPORT TYPES .... **34~~30~~**
20.1 Vertex Shading .... 34~~30~~

20.2 Pixel Shading .... 34~~30~~
21. SPECIAL INTERPOLATION MODES .... 35~~31~~
21.1 Real time commands .... 35~~31~~
21.2 Sprites/ XY screen coordinates/ FB information .... 35~~31~~
21.3 Auto generated counters .... 35~~31~~
21.3.1 Vertex shaders .... 35~~32~~
21.3.2 Pixel shaders .... 35~~32~~
22. STATE MANAGEMENT .... 36~~32~~
22.1 Parameter cache synchronization .... 36~~32~~
23. XY ADDRESS IMPORTS .... 36~~32~~
23.1 Vertex indexes imports .... 36~~33~~
24. REGISTERS .... 37~~33~~
24.1 Control .... 37~~33~~
24.2 Context .... 37~~33~~
25. DEBUG REGISTERS .... 38~~34~~
25.1 Context .... 38~~34~~
26. INTERFACES .... 38~~34~~
26.1 External Interfaces .... 38~~34~~
26.1.1 SC to SQ : IJ Control bus .... 38~~34~~
26.1.2 SQ to SP: Interpolator bus .... 39~~35~~
26.1.3 SQ to SP: Parameter Cache Read control bus .... 39~~35~~
26.1.4 SQ to SX: Parameter Cache Mux control Bus .... 40~~36~~
26.1.5 SQ to SP: Staging Register Data .... 40~~36~~
26.1.6 PA to SQ : Vertex interface .... 40~~36~~
26.1.7 SQ to CP: State report .... 44~~39~~
26.1.8 SQ to SX: Control bus .... 44~~39~~
26.1.9 SX to SQ : Output file control .... 44~~39~~
26.1.10 SQ to TP: Control bus .... 44~~39~~
26.1.11 TP to SQ: Texture stall .... 45~~40~~
26.1.12 SQ to SP: Texture stall .... 45~~40~~
26.1.13 SQ to SP: GPR, Parameter cache control and auto counter .... 45~~40~~
26.1.14 SQ to SPx: Instructions .... 46~~41~~
26.1.15 SP to SQ: Constant address load .... 47~~41~~
26.1.16 SQ to SPx: constant broadcast .... 47~~41~~
26.1.17 SP0 to SQ: Kill vector load .... 47~~42~~
26.1.18 SQ to CP: RBBM bus .... 47~~42~~
26.1.19 CP to SQ: RBBM bus .... 47~~42~~
27. EXAMPLES OF PROGRAM EXECUTIONS .... 48~~42~~
27.1.1 Sequencer Control of a Vector of Vertices .... 48~~42~~
27.1.2 Sequencer Control of a Vector of Pixels .... 49~~43~~
27.1.3 Notes .... 50~~44~~

28. OPEN ISSUES 5044
~~1. OVERVIEW .... 6~~
~~1.1 Top Level Block Diagram .... 8~~
~~1.2 Data Flow graph .... 10~~
~~1.3 Control Graph .... 11~~
~~2. INTERPOLATED DATA BUS .... 11~~
~~3. INSTRUCTION STORE .... 14~~
~~4. SEQUENCER INSTRUCTIONS .... 16~~
~~5. CONSTANT STORES .... 16~~
~~5.1 Memory organizations .... 16~~
~~5.2 Management of the re-mapping tables .... 16~~
~~5.2.1 Dirty bits .... 18~~
~~5.2.2 Free List Block .... 18~~
~~5.2.3 De-allocate Block .... 18~~
~~5.2.4 Operation of Incremental model .... 18~~
~~5.3 Constant Store Indexing .... 19~~
~~5.4 Real Time Commands .... 19~~
~~6. LOOPING AND BRANCHES .... 20~~
~~6.1 The controlling state. .... 20~~
~~6.2 The Control Flow Program .... 20~~
~~6.3 Data dependant predicate instructions .... 23~~
~~6.4 HW Detection of PV,PS .... 23~~
~~6.5 Register file indexing .... 23~~
~~6.6 Predicated Instruction support for Texture clauses .... 24~~
~~6.7 Debugging the Shaders .... 24~~
~~6.7.1 Method 1: Debugging registers .... 24~~
~~6.7.2 Method 2: Exporting the values in the GPRs (12) .... 25~~
~~7. PIXEL KILL MASK .... 25~~
~~8. MULTIPASS VERTEX SHADERS (HOS) .... 25~~
~~9. REGISTER FILE ALLOCATION .... 25~~
~~10. FETCH ARBITRATION .... 26~~
~~11. ALU ARBITRATION .... 26~~
~~12. HANDLING STALLS .... 27~~
~~13. CONTENT OF THE RESERVATION STATION FIFOS .... 27~~
~~14. THE OUTPUT FILE .... 27~~
~~15. IJ FORMAT .... 27~~
~~15.1 Interpolation of constant attributes .... 28~~
~~16. THE PARAMETER CACHE .... 30~~
~~17. VERTEX POSITION EXPORTING .... 30~~
~~18. EXPORTING ARBITRATION .... 30~~
~~19. EXPORT TYPES .... 30~~
~~19.1 Vertex Shading .... 30~~
~~19.2 Pixel Shading .... 30~~
~~20. SPECIAL INTERPOLATION MODES .... 31~~
~~20.1 Real time commands .... 31~~
~~20.2 Sprites/ XY screen coordinates/ FB information .... 31~~
~~20.3 Auto generated counters .... 31~~

~~20.3.1 Vertex shaders 32~~
~~20.3.2 Pixel shaders 32~~
~~21. STATE MANAGEMENT 32~~
~~21.1 Parameter cache synchronization 32~~
~~22. XY ADDRESS IMPORTS 32~~
~~22.1 Vertex indexes imports 33~~
~~23. REGISTERS 33~~
~~23.1 Control 33~~
~~23.2 Context 33~~
~~24. DEBUG REGISTERS 34~~
~~24.1 Context 34~~
~~25. INTERFACES 34~~
~~25.1 External Interfaces 34~~
~~25.1.1 SC to SQ : IJ Control bus 34~~
~~25.1.2 SQ to SP: Interpolator bus 35~~
~~25.1.3 SQ to SP: GPR Input Mux select 35~~
~~25.1.4 SQ to SP: Parameter Cache Read control bus 36~~
~~25.1.5 SQ to SX: Parameter Cache Mux control Bus 36~~
~~25.1.6 SQ to SP: Staging Register Data 36~~
~~25.1.7 PA to SQ : Vertex interface 36~~
~~25.1.8 SQ to CP: State report 40~~
~~25.1.9 SQ to SX: Control bus 40~~
~~25.1.10 SX to SQ : Output file control 40~~
~~25.1.11 SQ to TP: Control bus 40~~
~~25.1.12 TP to SQ: Texture stall 41~~
~~25.1.13 SQ to SP: Texture stall 41~~
~~25.1.14 SQ to SP: GPR and Parameter cache control 41~~
~~25.1.15 SQ to SPx: Instructions 42~~
~~25.1.16 SP to SQ: Constant address load 42~~
~~25.1.17 SQ to SPx: constant broadcast 42~~
~~25.1.18 SP0 to SQ: Kill vector load 43~~
~~25.1.19 SQ to CP: RBBM bus 43~~
~~25.1.20 CP to SQ: RBBM bus 43~~
~~26. EXAMPLES OF PROGRAM EXECUTIONS 43~~
~~26.1.1 Sequencer Control of a Vector of Vertices 43~~
~~26.1.2 Sequencer Control of a Vector of Pixels 44~~
~~26.1.3 Notes 45~~
~~27. OPEN ISSUES 45~~

Revision Changes:

Rev 0.1 (Laurent Lefebvre) Date: May 7, 2001	First draft.
Rev 0.2 (Laurent Lefebvre) Date : July 9, 2001	Changed the interfaces to reflect the changes in the SP. Added some details in the arbitration section.
Rev 0.3 (Laurent Lefebvre) Date : August 6, 2001	Reviewed the Sequencer spec after the meeting on August 3, 2001.
Rev 0.4 (Laurent Lefebvre) Date : August 24, 2001	Added the dynamic allocation method for register file and an example (written in part by Vic) of the flow of pixels/vertices in the sequencer.
Rev 0.5 (Laurent Lefebvre) Date : September 7, 2001	Added timing diagrams (Vic)
Rev 0.6 (Laurent Lefebvre) Date : September 24, 2001	Changed the spec to reflect the new R400 architecture. Added interfaces.
Rev 0.7 (Laurent Lefebvre) Date : October 5, 2001	Added constant store management, instruction store management, control flow management and data dependant predication.
Rev 0.8 (Laurent Lefebvre) Date : October 8, 2001	Changed the control flow method to be more flexible. Also updated the external interfaces.
Rev 0.9 (Laurent Lefebvre) Date : October 17, 2001	Incorporated changes made in the 10/18/01 control flow meeting. Added a NOP instruction, removed the conditional_execute_or_jump. Added debug registers.
Rev 1.0 (Laurent Lefebvre) Date : October 19, 2001	Refined interfaces to RB. Added state registers.
Rev 1.1 (Laurent Lefebvre) Date : October 26, 2001	Added SEQ→SP0 interfaces. Changed delta precision. Changed VGT→SP0 interface. Debug Methods added.
Rev 1.2 (Laurent Lefebvre) Date : November 16, 2001	Interfaces greatly refined. Cleaned up the spec.
Rev 1.3 (Laurent Lefebvre) Date : November 26, 2001	Added the different interpolation modes.
Rev 1.4 (Laurent Lefebvre) Date : December 6, 2001	Added the auto incrementing counters. Changed the VGT→SQ interface. Added content on constant management. Updated GPRs.
Rev 1.5 (Laurent Lefebvre) Date : December 11, 2001	Removed from the spec all interfaces that weren't directly tied to the SQ. Added explanations on constant management. Added PA→SQ synchronization fields and explanation.
Rev 1.6 (Laurent Lefebvre) Date : January 7, 2002	Added more details on the staging register. Added detail about the parameter caches. Changed the call instruction to a Conditionnal_call instruction. Added details on constant management and updated the diagram.
Rev 1.7 (Laurent Lefebvre) Date : February 4, 2002	Added Real Time parameter control in the SX interface. Updated the control flow section.

1. Overview

The sequencer is based on the R300 design. It chooses two ALU clauses and a fetch clause to execute, and executes all of the instructions in a clause before looking for a new clause of the same type. Two ALU clauses are executed interleaved to hide the ALU latency. Each vector will have eight fetch and eight ALU clauses, but clauses do not need to contain instructions. A vector of pixels or vertices ping-pongs along the sequencer FIFO, bouncing from fetch reservation station to alu reservation station. A FIFO exists between each reservation stage, holding up vectors until the vector currently occupying a reservation station has left. A vector at a reservation station can be chosen to execute. The sequencer looks at all eight alu reservation stations to choose an alu clause to execute and all eight fetch stations to choose a fetch clause to execute. The arbitrator will give priority to clauses/reservation stations closer to the bottom of the pipeline. It will not execute an alu clause until the fetch fetches initiated by the previous fetch clause have completed. There are two separate sets of reservation stations, one for pixel vectors and one for vertices vectors. This way a pixel can pass a vertex and a vertex can pass a pixel.

To support the shader pipe the sequencer also contains the shader instruction cache, constant store, control flow constants and texture state. The four shader pipes also execute the same instruction thus there is only one sequencer for the whole chip.

The sequencer first arbitrates between vectors of 64 vertices that arrive directly from primitive assembly and vectors of 16 quads (64 pixels) that are generated in the scan converter.

The vertex or pixel program specifies how many GPRs it needs to execute. The sequencer will not start the next vector until the needed space is available in the GPRs.

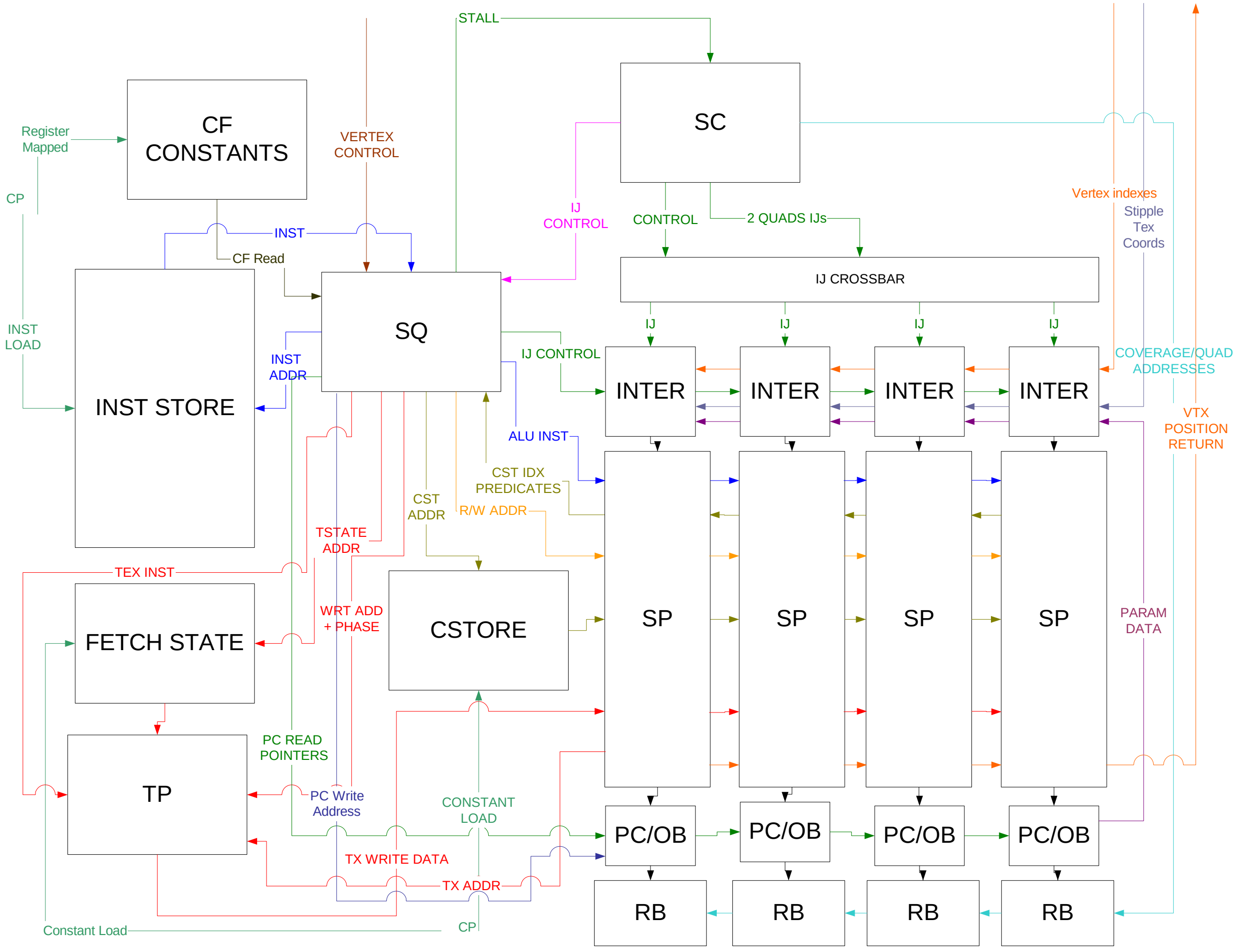
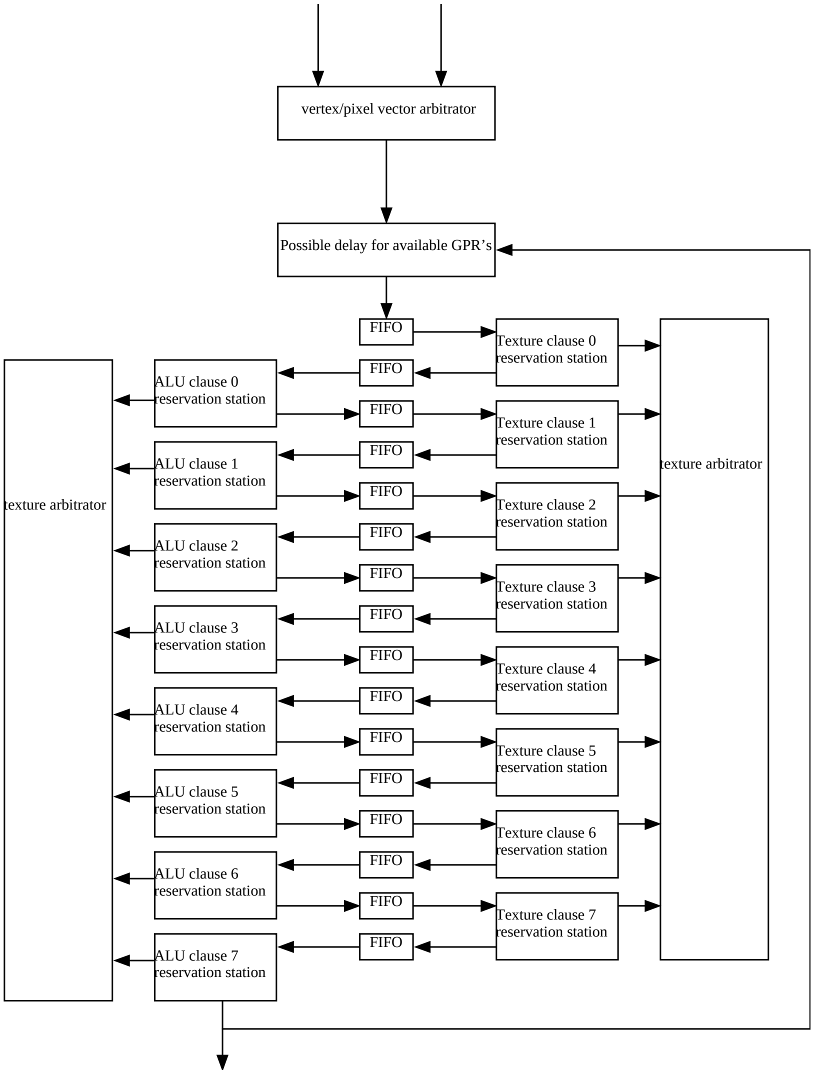


Figure 1: General Sequencer overview

1.1 Top Level Block Diagram



vertex/pixel vector arbitrator
Possible delay for available GPR's
FIFO
texture arbitrator
ALU clause 0 reservation station
ALU clause 1 reservation station
ALU clause 2 reservation station
ALU clause 3 reservation station
ALU clause 4 reservation station
ALU clause 5 reservation station
ALU clause 6 reservation station
ALU clause 7 reservation station
Texture clause 0 reservation station
Texture clause 1 reservation station
Texture clause 2 reservation station
Texture clause 3 reservation station
Texture clause 4 reservation station
Texture clause 5 reservation station
Texture clause 6 reservation station
Texture clause 7 reservation station
texture arbitrator

Figure 2: Reservation stations and arbiters

There are two sets of the above figure, one for vertices and one for pixels.

Depending on the arbitration state, the sequencer will either choose a vertex or a pixel packet. The control packet consists of 3 bits of state, 7 bits for the base address of the Shader program and some information on the coverage to determine fetch LOD plus other various small state bits.

On receipt of a packet, the input state machine (not pictured but just before the first FIFO) allocated enough space in the GPRs to store the interpolated values and temporaries. Following this, the barycentric coordinates (and XY screen position if needed) are sent to the ~~interpolator which~~interpolator, which will use them to interpolate the parameters and place the results into the GPRs. Then, the input state machine stacks the packet in the first FIFO.

On receipt of a command, the level 0 fetch machine issues a fetch request to the TP and corresponding GPR address for the fetch address (ta). A small command (tcmd) is passed to the fetch system identifying the current level number (0) as well as the GPR write address for the fetch return data. One fetch request is sent every 4 clocks causing the texturing of sixteen 2x2s worth of data (or 64 vertices). Once all the requests are sent the packet is put in FIFO 1.

Upon receipt of the return data, the fetch unit writes the data to the register file using the write address that was provided by the level 0 fetch machine and sends the clause number (0) to the level 0 fetch state machine to signify that the write is done and thus the data is ready. Then, the level 0 fetch machine increments the counter of FIFO 1 to signify to the ALU 0 that the data is ready to be processed.

On receipt of a command, the level 0 ALU machine first decrements the input FIFO 1 counter and then issues a complete set of level 0 shader instructions. For each instruction, the ALU state machine generates 3 source addresses, one destination address and an instruction. Once the last instruction has been issued, the packet is put into FIFO 2.

There will always be two active ALU clauses at any given time (and two arbiters). One arbiter will arbitrate over the odd instructions (4 clocks cycles) and the other one will arbitrate over the even instructions (4 clocks cycles). The only constraints between the two arbiters is that they are not allowed to pick the same clause number as the other one is currently working on if the packet is not of the same type (render state).

If the packet is a vertex packet, upon reaching ALU clause 3, it can export the position if the position is ready. So the arbiter must prevent ALU clause 3 to be selected if the positional buffer is full (or can't be accessed). Along with the positional data, if needed the sprite size and/or edge flags can also be sent.

A special case is for multipass vertex shaders, which can export 12 parameters per last 6 clauses to the output buffer. If the output buffer is full or doesn't have enough space the sequencer will prevent such a vertex group to enter an exporting clause.

Multipass pixel shaders can export 12 parameters to memory from the last clause only (7).

All other clauses process in the same way until the packet finally reaches the last ALU machine (7).

Only one pair of interleaved ALU state machines may have access to the register file address bus or the instruction decode bus at one time. Similarly, only one fetch state machine may have access to the register file address bus at one time. Arbitration is performed by three arbiter blocks (two for the ALU state machines and one for the fetch state machines). The arbiters always favor the higher number state machines, preventing a bunch of half finished jobs from clogging up the register files.

1.2 Data Flow graph (SP)

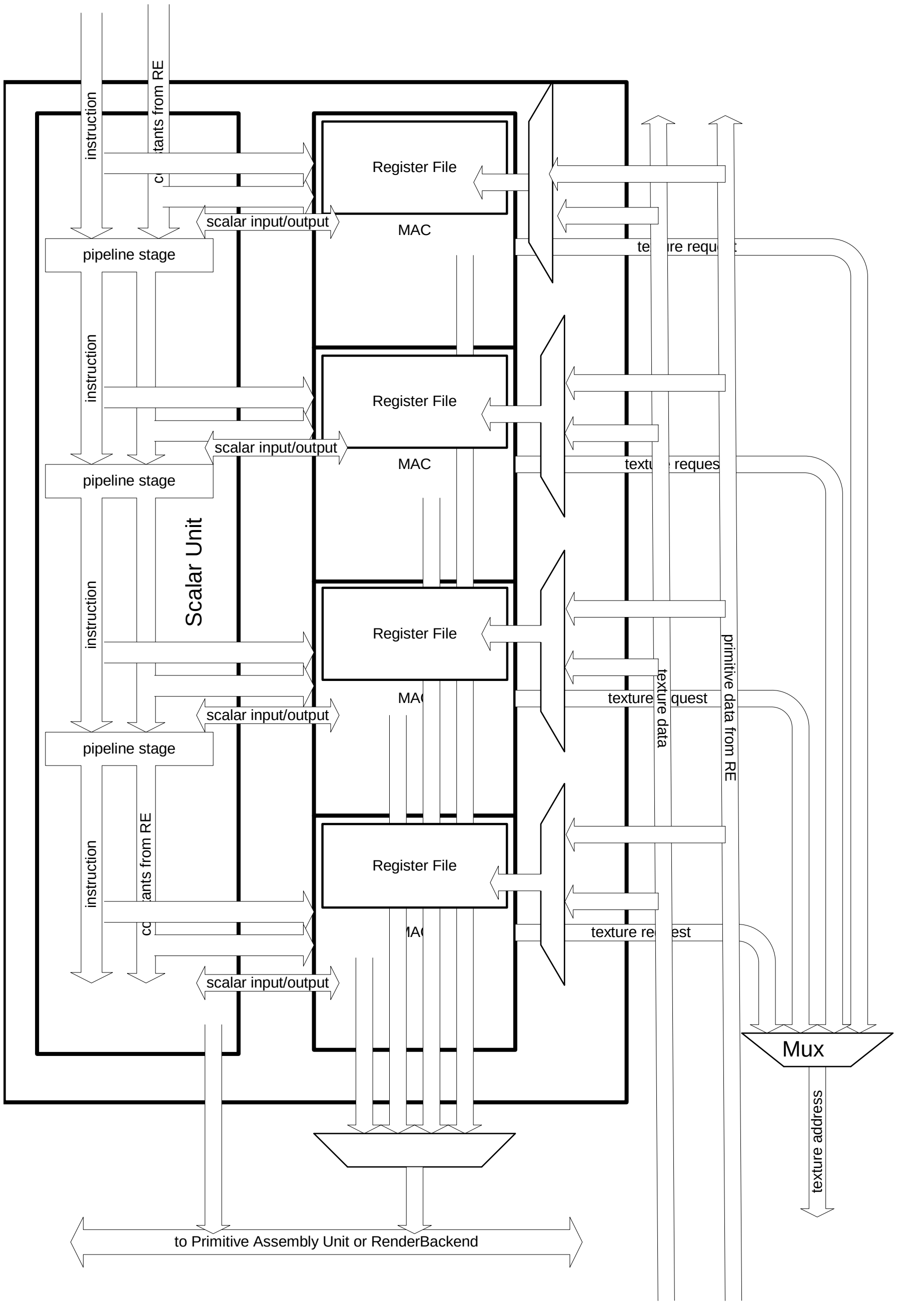


Figure 3: The shader Pipe

The gray area represents blocks that are replicated 4 times per shader pipe (16 times on the overall chip).

1.3 Control Graph

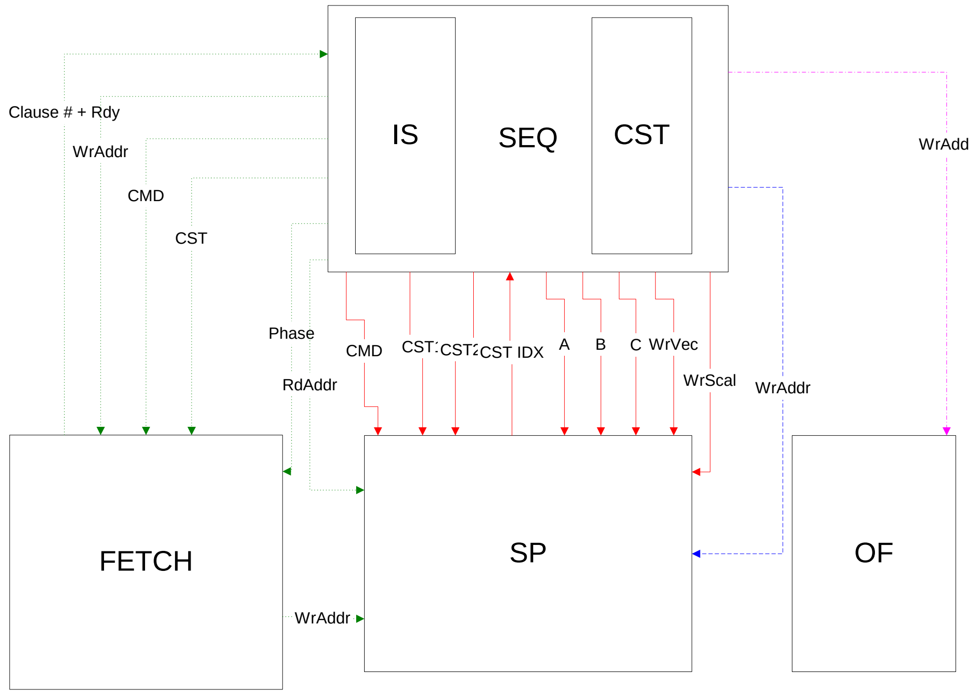


Figure 4: Sequencer Control interfaces

In green is represented the Fetch control interface, in red the ALU control interface, in blue the Interpolated/Vector control interface and in purple is the output file control interface.

2. Interpolated data bus

The interpolators contain an IJ buffer to pack the information as much as possible before writing it to the register file.

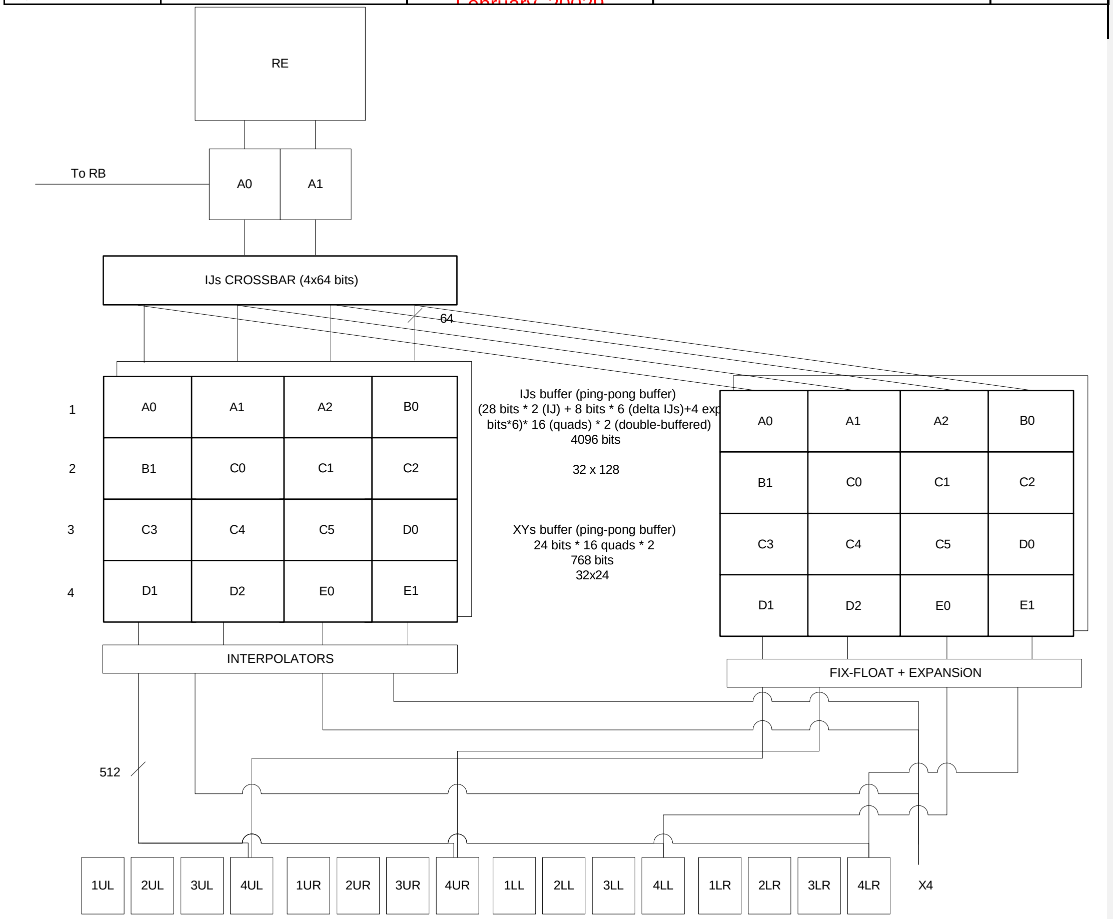


Figure 5: Interpolation buffers

WRITES

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21	T22	T23
SP 0	A0	A0	XY A0	B1	B1	XY B1	C3	C3	XY C3				D1	D1	XY D1									
SP 1	A1	A1	XY A1				C0	C0	XY C0	C4	C4	XY C4	D2	D2	XY D2									
SP 2	A2	A2	XY A2				C1	C1	XY C1	C5	C5	XY C5				E0	E0	XY E0						
SP 3				B0	B0	XY B0	C2	C2	XY C2				D0	D0	XY D0	E1	E1	XY E1						

READS

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21	T22	T23
SP 0	XY 0-3	XY 16-19	XY 32-35	XY 48-51	A0	B1	C3	D1					A0	B1	C3	D1					V 0-3	V 16-19	V 32-35	V 48-51
SP 1	XY 4-7	XY 20-23	XY 36-39	XY 52-55	A1		C4	D2		C0			A1		C4	D2		C0			V 4-7	V 20-23	V 36-39	V 52-55
SP 2	XY 8-11	XY 24-27	XY 40-43	XY 56-59	A2		C5			C1		E0	A2		C5			C1		E0	V 8-11	V 24-27	V 40-43	V 56-59
SP 3	XY 12-15	XY 28-31	XY 44-47	XY 60-63					B0	C2	D0	E1					B0	C2	D0	E1	V 12-15	V 28-31	V 44-47	V 60-63
	XY				P1								P2								VTX			

Figure 6: Interpolation timing diagram

Above is an example of a tile the sequencer might receive from the SC. The write side is how the data get stacked into the XY and IJ buffers, the read side is how the data is passed to the GPRs. The IJ information is packed in the IJ buffer 4 quads at a time or two clocks. The sequencer allows at any given time as many as four quads to interpolate a parameter. They all have to come from the same primitive. Then the sequencer controls the write mask to the GPRs to write the valid data in.

{ISSUE : Do we do the center + centroid approach using both IJ buffers?}

3. Instruction Store

There is going to be only one instruction store for the whole chip. It will contain 4096 instructions of 96 bits each.

It is likely to be a 1 ~~-~~port memory; we use ~~-~~1 clock to load the ALU instruction, 1 clocks to load the Fetch instruction, 1 clock to load 2 control flow instructions and 1 clock to write instructions.

The instruction store is loaded by the CP thru the register mapped registers.

The next picture shows the various modes the CP can load the memory. The Sequencer has to keep track of the loading modes in order to wrap around the correct boundaries. The wrap-~~-~~around points are arbitrary and they are specified in the VS_BASE and PIX_BASE control registers. The VS_BASE and PS_BASE context registers are used to specify for each context where its shader is in the instruction memory.

For the Real time commands the story is quite the same but for some small differences. There are no wrap-~~-~~around points for real time so the driver must be careful not to overwrite regular shader data. The shared code (shared subroutines) uses the same path as real time.

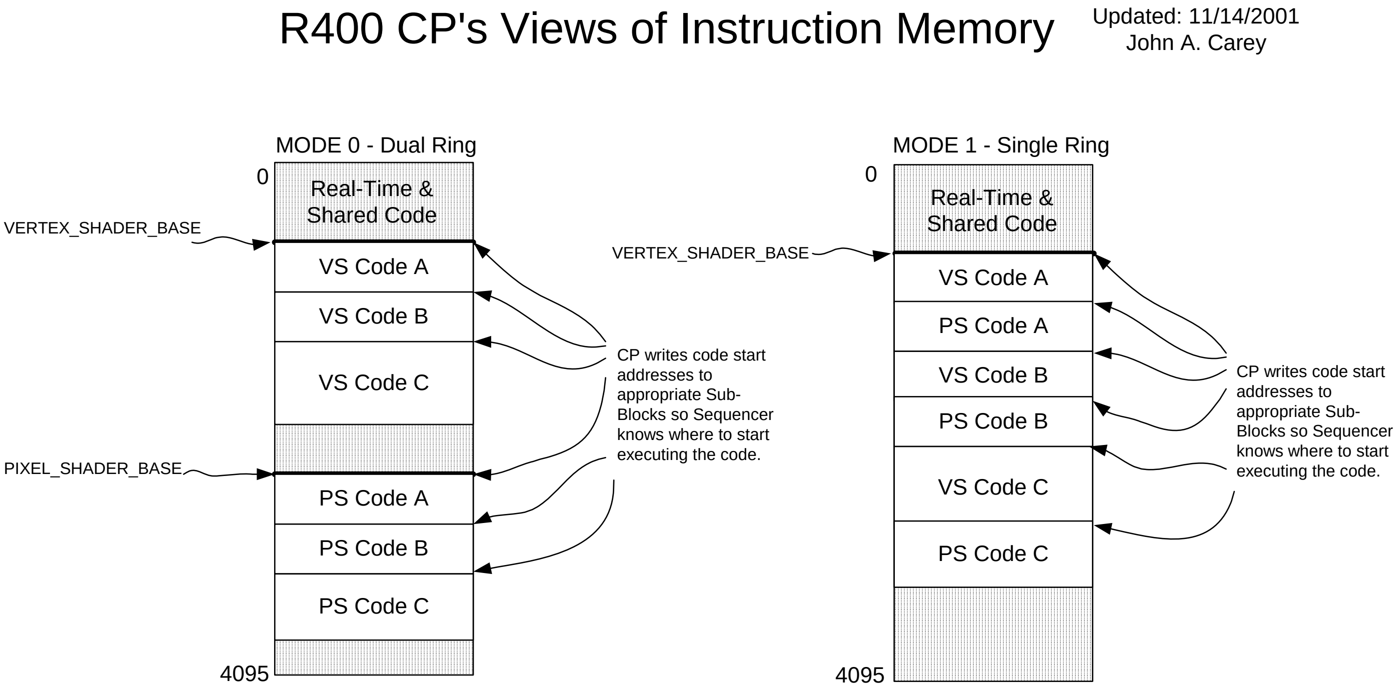


Figure 7: The CP's view of the instruction memory

4. Sequencer Instructions

All control flow instructions and move instructions are handled by the sequencer only. The ALUs will perform NOPs during this time (MOV PV,PV, PS,PS) if they have nothing else to do.

5. Constant Stores

5.1 Memory organizations

A likely size for the ALU constant store is 1024x128 bits. The read BW from the ALU constant store is 128 bits/clock and the write bandwidth is 32 bits/clock (directed by the CP bus size not by memory ports).

The maximum logical size of the constant store for a given shader is 256 constants. Or 512 for the pixel/vertex shader pair. The size of the re-mapping table is 128 lines (each line addresses 4 constants). The write granularity is 4 constants or 512 bits. It takes 16 clocks to write the four constants. Real time requires 256 lines in the physical memory (this is physically register mapped).

The texture state is also kept in a similar memory. The size of this memory is ~~192x128~~128x192 bits. The memory thus holds 128 texture states (192 bits per state). The logical size exposes 32 different states total, which are going to be shared between the pixel and the vertex shader. The size of the re-mapping table to for the texture state memory is ~~16~~ 32 lines (each line addresses ~~2~~ 1 texture state lines in the real memory). The CP write granularity is ~~2~~ 1 textur~~e~~e state lines (or ~~384~~ 192 bits). The driver sends 512 bits but the CP ignores the top ~~128~~ 320 bits. It thus takes ~~12~~ 6 clocks to write the ~~two~~ texture state~~s~~. Real time requires 32 lines in the physical memory (this is physically register mapped).

The control flow constant memory doesn’t sit behind a renaming table. It is register mapped and thus the driver must reload its content each time there is a state change. Its size is 320*32 because it must hold 8 copies of the 32 dwords of control flow constants and the loop construct constants must be aligned.

The constant re-mapping tables for texture state and ALU constants are logically register mapped for regular mode and physically register mapped for RT operation.

5.2 Management of the re-mapping tables

5.2.1 *R400 Constant management*

Formatted: Bullets and Numbering

The sequencer is responsible to manage two re-mapping tables (one for the constant store and one for the texture state). On a state change (by the driver), the sequencer will broadside copy the contents of its re-mapping tables to a new one. We have 8 different re-mapping tables we can use concurrently.

The constant memory update will be incremental, the driver only need to update the constants that actually changed between the two state changes.

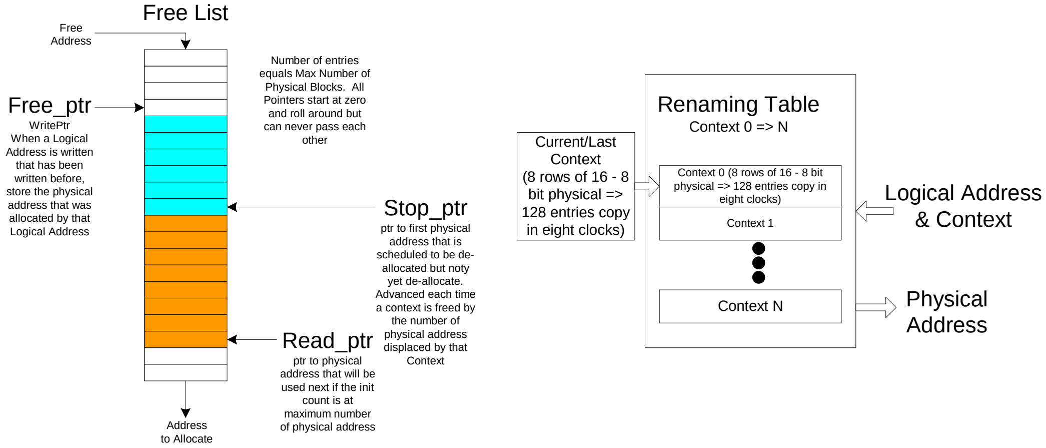
For this model to work in its simplest form, the requirement is that the physical memory MUST be at least twice as large as the logical address space + the space allocated for Real Time. In our case, since the logical address space is 512 and the reserved RT space can be up to 256 entries, the memory must be of sizes ~~1024~~ 1280 and above. Similarly the size of the texture store must be of 32*2+32 = 96 entries and above.

~~5.2.1~~5.2.2 *Proposal for R400LE constant management*

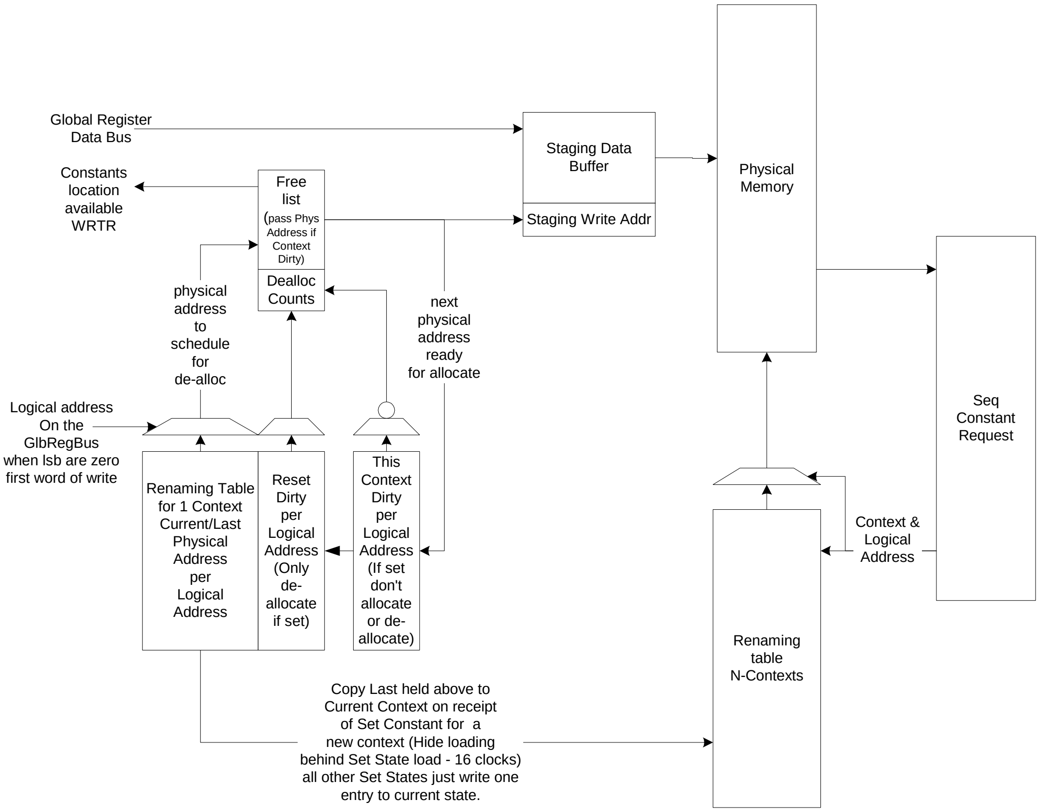
Formatted: Bullets and Numbering

To make this scheme work with only 512+256 = 768 entries, upon reception of a CONTROL packet of state + 1, the sequencer would check for SQ_IDLE and PA_IDLE and if both are idle will erase the content of state to replace it with the new state (this is depicted in Figure 9: De-allocation mechanism~~Figure 9: De-allocation mechanism~~~~Figure 2: De-allocation mechanism~~ ~~Figure 1: Dealocation mechanism~~). Note that in the case a state is cleared a value of 0 is written to the corresponding de-allocation counter location so that when the SQ is going to report a state change, nothing will be de-allocated upon the first report.

The second path sets all context dirty bits that were used in the current state to 1 (thus allowing the new state to reuse these physical addresses if needed). ~~Be careful to set only those bits that the CURENT STATE IS USING (if for example the current state uses only 64 constants we set only lines 0 thru 15 to 1). This is ok to do so because the blocks are idle and thus the context has finished drawing.~~



Free List
Free Address
Free_ptr
WritePtr When a Logical Address is written that has been written before, store the physical address that was allocated by that Logical Address
Number of entries equals Max Number of Physical Blocks. All Pointers start at zero and roll around but can never pass each other
Stop_ptr
ptr to first physical address that is scheduled to be de-allocated but noty yet de-allocate. Advanced each time a context is freed by the number of physical address displaced by that Context
Read_ptr
ptr to physical address that will be used next if the init count is at maximum number of physical address
Address to Allocate
Current/Last Context (8 rows of 16 - 8 bit physical => 128 entries copy in eight clocks)
Renaming Table
Context 0 => N
Context 0 (8 rows of 16 - 8 bit physical => 128 entries copy in eight clocks)
Context 1
Context N
Logical Address & Context
Physical Address



Global Register Data Bus
Staging Data Buffer
Physical Memory
Constants location available WRTR
Free list (pass Phys Address if Context Dirty)
Dealloc Counts
Staging Write Addr
physical address to schedule for de-alloc
next physical address ready for allocate
Logical address On the GlbRegBus when lsb are zero first word of write
Renaming Table for 1 Context Current/Last Physical Address per Logical Address
Reset Dirty per Logical Address (Only de-allocate if set)
This Context Dirty per Logical Address (If set don't allocate or de-allocate)
Seq Constant Request
Context & Logical Address
Renaming table N-Contexts
Copy Last held above to Current Context on receipt of Set Constant for a new context (Hide loading behind Set State load - 16 clocks) all other Set States just write one entry to current state.

Figure 8: Constant management

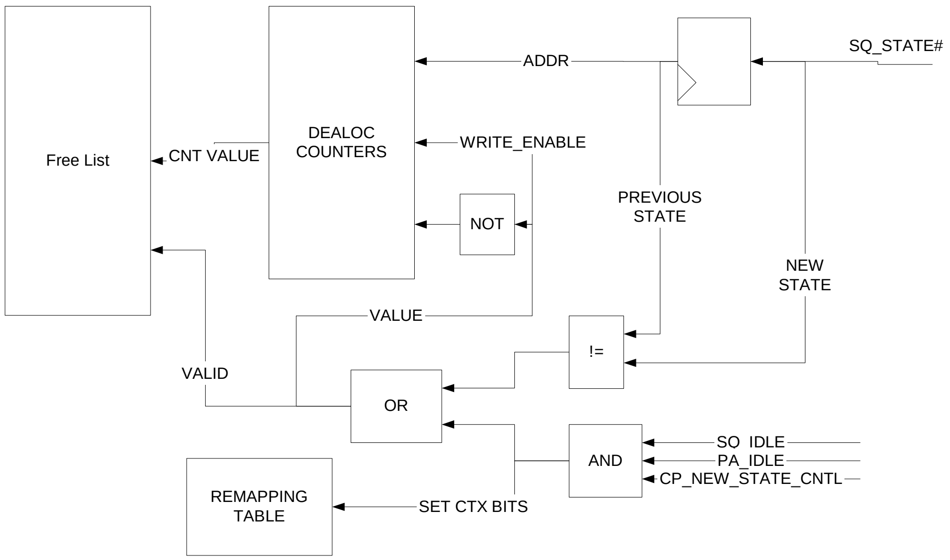


Figure 9: De-allocation mechanism for R400LE

~~5.2.1~~5.2.3 *Dirty bits*

Two sets of dirty bits will be maintained per logical address. The first one will be set to zero on reset and set when the logical address is addressed. The second one will be set to zero when-ever a new context is written and set for each address written while in this context. The reset dirty is not set, then writing to that logical address will not require de-allocation of whatever address stored in the renaming table. If it is set and the context dirty is not set, then the physical address store needs to be de-allocated and a new physical address is necessary to store the incoming data. If they are both set, then the data will be written into the physical address held in the renaming for the current logical address. No de-allocation or allocation takes place. This will happen when the driver does a set constant twice to the same logical address between context changes. NOTE: It is important to detect and prevent this, failure to do it will allow multiple writes to allocate all physical memory and thus hang because a context will not fit for rendering to start and thus free up space.

Formatted: Bullets and Numbering

~~5.2.2~~5.2.4 *Free List Block*

Formatted: Bullets and Numbering

A free list block that would consist of a counter (called the IFC or Initial Free Counter) that would reset to zero and incremented every time a chunk of physical memory is used until they have all been used once. This counter would be checked each time a physical block is needed, and if the original ones have not been used up, us a new one, else check the free list for an available physical block address. The count is the physical address for when getting a chunk from the counter.

Storage of a free list big enough to store all physical block addresses.

Maintain three pointers for the free list that are reset to zero. The first one we will call write_ptr. This pointer will identify the next location to write the physical address of a block to be de-allocated. Note: we can never free more physical memory locations than we have. Once recording address the pointer will be incremented to walk the free list like a ring.

The second pointer will be called stop_ptr. The stop_ptr pointer will be advanced by the number of address chunks de-allocates when a context finishes. The address between the stop_ptr and write_ptr cannot be reused because they are still in use. But as soon as the context using then is dismissed the stop_ptr will be advanced.

The third pointer will be called read_ptr. This pointer will point will point to the next address that can be used for allocation as long as the read_ptr does not equal the stop_ptr and the IFC is at its maximum count.

~~5.2.3~~5.2.5 *De-allocate Block*

Formatted: Bullets and Numbering

This block will maintain a free physical address block count for each context. While in current context, a count shall be maintained specifying how many blocks were written into the free list at the write_ptr pointer. This count will be reset upon reset or when this context is active on the back and different than the previous context. It is actually a count of blocks in the previous context that will no longer be used. This count will be used to advance the write_ptr pointer to make available the set of physical blocks freed when the previous context was done. This allows the discard or de-allocation of any number of blocks in one clock.

~~5.2.4~~5.2.6 *Operation of Incremental model*

Formatted: Bullets and Numbering

The basic operation of the model would start with the write_ptr, stop_ptr, read_ptr pointers in the free list set to zero and the free list counter is set to zero. Also all the dirty bits and the previous context will be initialized to zero. When the first set constants happen, the reset dirty bit will not be set, so we will allocate a physical location from the free list counter because its not at the max value. The data will be written into physical address zero. Both the additional copy of the renaming table and the context zeros of the big renaming table will be updated for the logical address that was written by set start with physical address of 0. This process will be repeated for any logical address that are not dirty until the context changes. If a logical address is hit that has its dirty bits set while in the same context, both dirty bits would be set, so the new data will be over-written to the last physical address assigned for this logical address. When the first draw command of the context is detected, the previous context stored in the additional renaming table will be copied to the larger renaming table in the current (new) context location. Then the set constant logical address with be loaded with a new physical address during the copy and if the reset dirty was set, the physical address it replaced in the renaming table would be entered at the write_ptr pointer location on the free list and the write_ptr will be incremented. The de-allocation counter for the previous context (eight) will be incremented. This as set states come in for this context one of the following will happen:

1.) No dirty bits are set for the logical address being updated. A line will be allocated of the free-list counter or the free list at read_ptr pointer if read_ptr != to stop_ptr .
2.) Reset dirty set and Context dirty not set. A new physical address is allocated, the physical address in the renaming table is put on the free list at write_ptr and it is incremented along with the de-allocate counter for the last context.
3.) Context dirty is set then the data will be written into the physical address specified by the logical address.

This process will continue as long as set states arrive. This block will provide backpressure to the CP whenever he has not free list entries available (counter at max and stop_ptr == read_ptr). The command stream will keep a count of contexts of constants in use and prevent more than max constants contexts from being sent.

Whenever a draw packet arrives, the content of the re-mapping table is written to the correct re-mapping table for the context number. Also if the next context uses less constants than the current one all exceeding lines are moved to the free list to be de-allocated later. This happens in parallel with the writing of the re-mapping table to the correct memory.

Now preferable when the constant context leaves the last ALU clause it will be sent to this block and compared with the previous context that left. (Init to zero) If they differ than the older context will no longer be referenced and thus can be de-allocated in the physical memory. This is accomplished by adding the number of blocks freed this context to the stop_ptr pointer. This will make all the physical addresses used by this context available to the read_ptr allocate pointer for future allocation.

This device allows representation of multiple contexts of constants data with N copies of the logical address space. It also allows the second context to be represented as the first set plus some new additional data by just storing the delta's. It allows memory to be efficiently used and when the constants updates are small it can store multiple context. However, if the updates are large, less contexts will be stored and potentially performance will be degraded. Although it will still perform as well as a ring could in this case.

5.3 Constant Store Indexing

In order to do constant store indexing, the sequencer must be loaded first with the indexes (that come from the GPRs). There are 144 wires from the exit of the SP to the sequencer (9 bits pointers x 16 vertexes/clock). Since the data must pass thru the Shader pipe for the float to fixed conversion, there is a latency of 4 clocks (1 instruction)

between the time the sequencer is loaded and the time one can index into the constant store. The assembly will look like this

```
MOVA  R1.X,R2.X       // Loads the sequencer with the content of R2.X, also copies the content of R2.X into R1.X
NOP                   // latency of the float to fixed conversion
ADD   R3,R4,C0[R2.X]// Uses the state from the sequencer to add R4 to C0[R2.X] into R3
```

Note that we don't really care about what is in the brackets because we use the state from the MOVA instruction. R2.X is just written again for the sake of simplicity and coherency.

The storage needed in the sequencer in order to support this feature is 2*64*9 bits = 1152 bits.

5.4 Real Time Commands

The real time commands constants are written by the CP using the register mapped registers allocated for RT. It works is the same way than when dealing with regular constant loads BUT in this case the CP is not sending a logical address but rather a physical address and the reads are not passing thru the re-mapping table but are directly read from the memory. The boundary between the two zones is defined by the CONST_EO_RT control register. Similarly, for the fetch state, the boundary between the two zones is defined by the TSTATE_EO_RT control register.

5.5 Constant Waterfalling

In order to have a reasonable performance in the case of constant store indexing using the address register, we are going to have the possibility of using the physical memory port for read only. This way we can read 1 constant per clock and thus have a worst-case waterfall mode of 1 vertex per clock. There is a small synchronization issue related with this as we need for the SQ to make sure that the constants where actually written to memory (not only sent to the sequencer) before it can allow the first vector of pixels or vertices of the state to go thru the ALUs. To do so, the sequencer keeps 8 bits (one per render state) and sets the bits whenever the last render state is written to memory and clears the bit whenever a state is freed.

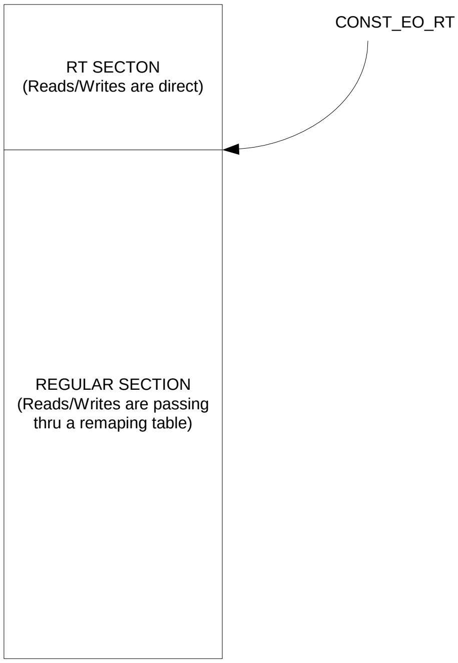

CONST_EO_RT
RT SECTON
(Reads/Writes are direct)
REGULAR SECTION
(Reads/Writes are passing thru a remaping table)

Figure 10: The instruction store

6. Looping and Branches

Loops and branches are planned to be supported and will have to be dealt with at the sequencer level. We plan on supporting constant loops and branches using a control program.

6.1 The controlling state.

The R400 controling state consists of:

Boolean[256:0]
Loop_count[7:0][31:0]
Loop_Start[7:0][31:0]
Loop_Step[7:0][31:0]

That is 256 Booleans and 32 loops.

We have a stack of 4 elements for nested calls of subroutines and 4 loop counters to allow for nested loops.

This state is available on a per shader program basis.

6.2 The Control Flow Program

Examples of control flow programs are located in the R400 programming guide document.

The basic model is as follows:

The render state defined the clause boundaries:

Vertex_shader_fetch[7:0][7:0]	// eight 8 bit pointers to the location where each clauses control program is located
Vertex_shader_alu[7:0][7:0]	// eight 8 bit pointers to the location where each clauses control program is located
Pixel_shader_fetch[7:0][7:0]	// eight 8 bit pointers to the location where each clauses control program is located
Pixel_shader_alu[7:0][7:0]	// eight 8 bit pointers to the location where each clauses control program is located

A pointer value of FF means that the clause doesn't contain any instructions.

The control program for a given clause is executed to completion before moving to another clause, (with the exception of the pick two nature of the alu execution). The control program is the only program aware of the clause boundaries.

The control program has eleven basic instructions:

Execute
Conditional_execute
Conditional_Execute_Predicates
Conditional_jump
Conditionnal_Call
Return
Loop_start
Loop_end
End_of_clause
Conditional_End_of_clause
NOP

Execute, causes the specified number of instructions in instruction store to be executed.
Conditional_execute checks a condition first, and if true, causes the specified number of instructions in instruction store to be executed.
Loop_start resets the corresponding loop counter to the start value on the first pass after it checks for the end condition and if met jumps over to a specified address.
Loop_end increments (decrements?) the loop counter and jumps back the specified number of instructions.

Conditionnal_Call jumps to an address and pushes the IP counter on the stack if the condition is met. On the return instruction, the IP is popped from the stack.
~~Conditional_execute_or_Jump executes a block of instructions or jumps to an address is the condition is not met.~~
Conditional_execute_Predicates executes a block of instructions if all bits in the predicate vectors meet the condition.
End_of_clause marks the end of a clause.
Conditional_End_of_clause marks the end of a clause if the condition is met.
Conditional_jumps jumps to an address if the condition is met.
NOP is a regular NOP

NOTE THAT ALL JUMPS MUST JUMP TO EVEN CFP ADDRESSES since there are two control flow instructions per memory line. Thus the compiler must insert NOPs where needed to align the jumps on even CFP addresses.

Also if the jump is logically bigger than pshader_cntl_size (or vshader_cntl_size) we break the program (clause) and set the debug registers. If an execute or conditional_execute is lower than cntl_size or bigger than size we also break the program (clause) and set the debug registers.

We have to fit instructions into 48 bits in order to be able to put two control flow instruction per line in the instruction store.

Note that whenever a field is marked as RESERVED, it is assumed that all the bits of the field are cleared (0).

Execute				
47	46… 42	41 … 24	23 … 12	11 … 0
Addressing	00001	RESERVED	Instruction count	Exec Address

Execute up to 4k instructions at the specified address in the instruction memory.

NOP		
47	46 … 42	41 … 0
Addressing	00010	RESERVED

This is a regular NOP.

Conditional_Execute							
47	46 … 42	41	40 … 33	32	31 … 24	23 … 12	11 … 0
Addressing	00011	RESERVED	Boolean address	Condition	RESERVED	Instruction count	Exec Address

If the specified Boolean (8 bits can address 256 Booleans) meets the specified condition then execute the specified instructions (up to 4k instructions)

Conditional_Execute_Predicates							
47	46 … 42	41 … 35	34 … 33	32	31 … 24	23 … 12	11 … 0
Addressing	00100	RESERVED	Predicate vector	Condition	RESERVED	Instruction count	Exec Address

Check the AND/OR of all current predicate bits. If AND/OR matches the condition execute the specified number of instructions. We need to AND/OR this with the kill mask in order not to consider the pixels that aren't valid.

Loop_Start				
47	46 … 42	41 … 17	16 … 12	11 … 0
Addressing	00101	RESERVED	loop ID	Jump address

Loop Start. Compares the loop iterator with the end value. If loop condition not met jump to the address. Forward jump only. Also computes the index value. The loop id must match between the start to end, and also indicates which control flow constants should be used with the loop.

Loop_End				
47	46 … 42	41 … 17	16 … 12	11 … 0
Addressing	00110~~1~~	RESERVED	loop ID	start address

Loop end. Increments the counter by one, compares the loop count with the end value. If loop condition met, continue, else, jump BACK to the start of the loop.

The way this is described does not prevent nested loops, and the inclusion of the loop id make this easy to do.

Conditionnal_Call						
47	46 … 42	41 … 35	34 … 33	32	31 … 12	11 … 0
Addressing	0~~1~~0111~~0~~ ~~00~~	RESERVED	Predicate vector	Condition	RESERVED	Jump address

If the condition is met, jumps to the specified address and pushes the control flow program counter on the stack.

Return		
47	46 … 42	41 … 0
Addressing	01000~~00~~ ~~1~~	RESERVED

Pops the topmost address from the stack and jumps to that address. If nothing is on the stack, the program will just continue to the next instruction.

Conditionnal_Jump							
47	46 … 42	41	40 … 33	32	31	30 … 12	11 … 0
Addressing	01001~~10~~	RESERVED	Boolean address	Condition	FW only	RESERVED	Jump address

If condition met, jumps to the address. FORWARD jump only allowed if bit 31 set. Bit 31 is only an optimization for the compiler and should NOT be exposed to the API.

Conditional_End_of_Clause					
47	46 … 42	41	40 … 33	32	31 … 0
Addressing	01010~~1~~	RESERVED	Boolean address	Condition	RESERVED

This is an optimization in the case of very short shaders (where the control flow instruction can't be hidden anymore and thus are not free. In this case, if the condition is met, the clause is ended, else we continue the execution of the clause.

End_of_Clause		
47	46 … 42	41 … 0
Addressing	01011~~0~~ ~~11~~	RESERVED

Marks the end of a clause.

To prevent infinite loops, we will keep ~~9 ~~9 bits loop ~~counters ~~iterators instead of 8 (we are only able to loop 256 times). If the counter goes higher than 255 then the loop_end or the loop_start instruction is going to break the loop and set the debug GPRs.

6.3 Data dependant predicate instructions

Data dependant conditionals will be supported in the R400. The only way we plan to support those is by supporting three vector/scalar predicate operations of the form:

PRED_SETE_# - similar to SETE except that the result is 'exported' to the sequencer.

PRED_SETNE_# - similar to SETNE except that the result is 'exported' to the sequencer.
PRED_SETGT_# - similar to SETGT except that the result is 'exported' to the sequencer
PRED_SETGTE_# - similar to SETGTE except that the result is 'exported' to the sequencer

For the scalar operations only we will also support the two following instructions:
PRED_SETE0_# – SETE0
PRED_SETE1_# – SETE1

The export is a single bit - 1 or 0 that is sent using the same data path as the MOVA instruction. The sequencer will maintain 4 sets of 64 bit predicate vectors (in fact 8 sets because we interleave two programs but only 4 will be exposed) and use it to control the write masking. This predicate is not maintained across clause boundaries. The # sign is used to specify which predicate set you want to use 0 thru 3.

Then we have two conditional execute bits. The first bit is a conditional execute “on” bit and the second bit tells us if we execute on 1 or 0. For example, the instruction:

P0_ADD_# R0,R1,R2

Is only going to write the result of the ADD into those GPRs whose predicate bit is 0. Alternatively, P1_ADD_# would only write the results to the GPRs whose predicate bit is set. The use of the P0 or P1 without precharging the sequencer with a PRED instruction is undefined.

{Issue: do we have to have a NOP between PRED and the first instruction that uses a predicate?}

6.4 HW Detection of PV,PS

Because of the control program, the compiler cannot detect statically dependant instructions. In the case of non-masked writes and subsequent reads the sequencer will insert uses of PV,PS as needed. This will be done by comparing the read address and the write address of consecutive instructions. For masked writes, the sequencer will insert NOPs wherever there is a dependant read/write.

The sequencer will also have to insert NOPs between PRED_SET and MOVA instructions and their uses.

6.5 Register file indexing

Because we can have loops in fetch clause, we need to be able to index into the register file in order to retrieve the data created in a fetch clause loop and use it into an ALU clause. The instruction will include the base address for register indexing and the instruction will contain these controls:

Bit7	Bit 6	
0	0	'absolute register'
0	1	'relative register'
1	0	'previous vector'
1	1	'previous scalar'

In the case of an absolute register we just take the address as is. In the case of a relative register read we take the base address and we add to it the loop_index and this becomes our new address that we give to the shader pipe.

The sequencer is going to keep a loop index computed as such:

Index = Loop_iterator*Loop_step + Loop_start.

~~The index is going to return 0 if it is out of the range.~~
We loop until loop_iterator = loop_count. Loop_step is a signed value [-128…127]. The computed index value is a 10 bit counter that is also signed. Its real range is [-256,256]. The tenth bit is only there so that we can provide an out of range value to the “indexing logic” so that it knows when the provided index is out of range and thus can make the necessary arrangements.

6.6 Predicated Instruction support for Texture clauses

For texture clauses, we support the following optimization: we keep 1 bit (thus 4 bits for the four predicate vectors) per predicate vector in the reservation stations. A value of 1 means that one ore more elements in the vector have a value of one (thus we have to do the texture fetches for the whole vector). A value of 0 means that no elements in the vector have his predicate bit set and we can thus skip over the texture fetch. **We have to make sure the invalid pixels aren't considered with this optimization.**

6.7 Debugging the Shaders

In order to be able to debug the pixel/vertex shaders efficiently, we provide 2 methods.

6.7.1 *Method 1: Debugging registers*

Current plans are to expose 2 debugging, or error notification, registers:
1. address register where the first error occurred
2. count of the number of errors

The sequencer will detect the following groups of errors:
- count overflow
~~-~~- constant indexing overflow
- register indexing overflow

Formatted: Bullets and Numbering

Compiler recognizable errors:
- jump errors
    relative jump address > size of the control flow program
    ~~relative jump address > length of the shader program~~
~~–~~- call stack
    call with stack full
    return with stack empty

~~With two of the errors, a~~A jump error ~~or a register overflow~~ will always cause the program to break. In this case, a break means that a clause will halt execution, but allowing further clauses to be executed.

With all the other errors, program can continue to run, potentially to worst-case limits. The program will only break if the DB_PROB_BREAK register is set.

If indexing outside of the ~~constant~~ constant or the register range, causing an overflow error, the hardware is specified to return the value with an index of 0. This could be exploited to generate error tokens, by reserving and initializing the 0th register (or constant) for errors.

{ISSUE : Interrupt to the driver or not?}

6.7.2 *Method 2: Exporting the values in the GPRs (12)*

The sequencer will have a count register and an address register for this mode and 3 bits per clause specifying the execution mode for each clause. The modes can be :
1) Normal
2) Debug Kill
3) Debug Addr + Count

Under the normal mode execution follows the normal course. Under the kill mode, all control flow instructions are executed but all normal shader instructions of the clause are replaced by NOPs. Only debug_export instructions of clause 7 will be executed under the debug kill setting. Under the other mode, normal execution is done until we reach an address specified by the address register and instruction count (useful for loops) specified by the count register. After we have hit the instruction n times (n=count) we switch the clause to the kill mode.

Under the debug mode (debug kill OR debug Addr + count), it is assumed that clause 7 is always exporting 12 debug vectors and that all other exports to the SX block (position, color, z, ect) will been turned off (changed into NOPs) by the sequencer (even if they occur before the address stated by the ADDR debug register).

7. Pixel Kill Mask

A vector of 64 bits is kept by the sequencer per group of pixels/vertices. Its purpose is to optimize the texture fetch requests and allow the shader pipe to kill pixels using the following instructions:

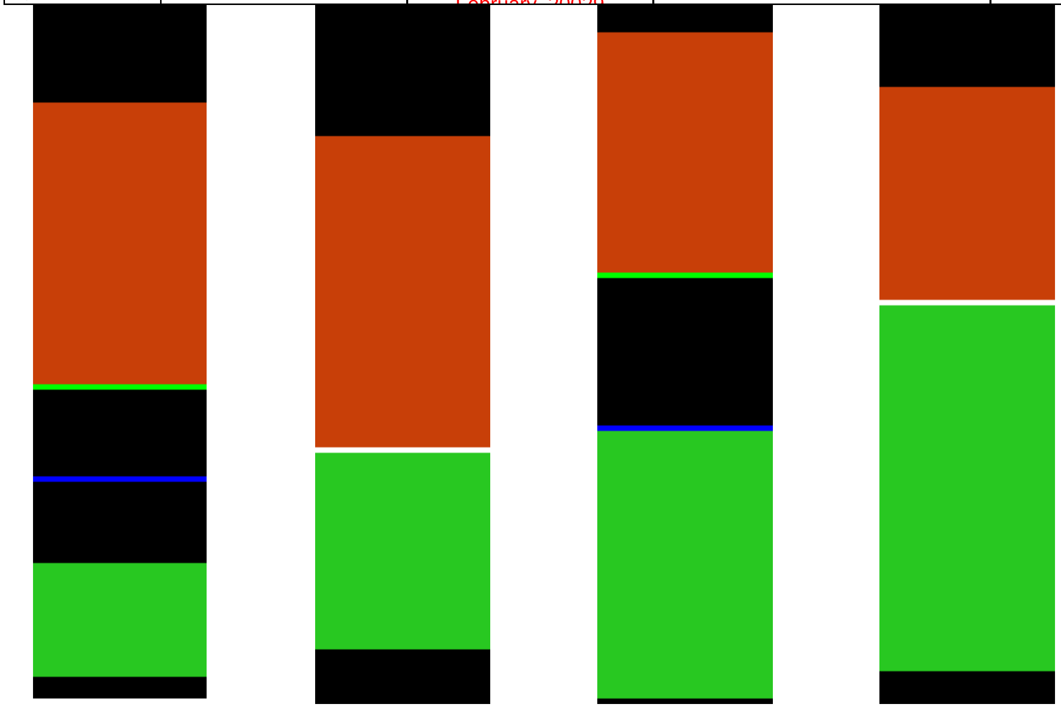
MASK_SETE
MASK_SETNE
MASK_SETGT
MASK_SETGTE

8. Multipass vertex shaders (HOS)

Multipass vertex shaders are able to export from the 6 last clauses but to memory ONLY.

9. Register file allocation

The register file allocation for vertices and pixels can either be static or dynamic. In both cases, the register file in managed using two round robins (one for pixels and one for vertices). In the dynamic case the boundary between pixels and vertices is allowed to move, in the static case it is fixed to 128-VERTEX_REG_SIZE for vertices and ~~256-~~ ~~VERTEX~~PIXEL_REG_SIZE for pixels.



Above is an example of how the algorithm works. Vertices come in from top to bottom; pixels come in from bottom to top. Vertices are in orange and pixels in green. The blue line is the tail of the vertices and the green line is the tail of the pixels. Thus anything between the two lines is shared. When pixels meets vertices the line turns white and the boundary is static until both vertices and pixels share the same “unallocated bubble”. Then the boundary is allowed to move again. The numbering of the GPRs starts from the bottom of the picture at index 0 and goes up to the top at index 127.

10. Fetch Arbitration

The fetch arbitration logic chooses one of the 8 potentially pending fetch clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute. Once chosen, the clause state machine will send one 2x2 fetch per clock (or 4 fetches in one clock every 4 clocks) until all the fetch instructions of the clause are sent. This means that there cannot be any dependencies between two fetches of the same clause.

The arbitrator will not wait for the fetches to return prior to selecting another clause for execution. The fetch pipe will be able to handle up to X(?) in flight fetches and thus there can be a fair number of active clauses waiting for their fetch return data.

11. ALU Arbitration

ALU arbitration proceeds in almost the same way than fetch arbitration. The ALU arbitration logic chooses one of the 8 potentially pending ALU clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute. There are two ALU arbiters, one for the even clocks and one for the odd clocks. For example, here is the sequencing of two interleaved ALU clauses (E and O stands for Even and Odd sets of 4 clocks):

Einst0 Oinst0 Einst1 Oinst1 Einst2 Oinst2 Einst0 Oinst3 Einst1 Oinst4 Einst2 Oinst0…
Proceeding this way hides the latency of 8 clocks of the ALUs. Also note that the interleaving also occurs across clause boundaries.

12. Handling Stalls

When the output file is full, the sequencer prevents the ALU arbitration logic from selecting the last clause (this way nothing can exit the shader pipe until there is place in the output file. If the packet is a vertex packet and the position buffer is full (POS_FULL) then the sequencer also prevents a thread from entering the exporting clause (3?). The sequencer will set the OUT_FILE_FULL signal n clocks before the output file is actually full and thus the ALU arbiter will be able read this signal and act accordingly by not preventing exporting clauses to proceed.

13. Content of the reservation station FIFOs

The reservation FIFOs contain the state of the vector of pixels and vertices. We have two sets of those: one for pixels, and one for vertices. They contain 3 bits of Render State 7 bits for the base address of the GPRs, some bits for LOD correction and coverage mask information in order to fetch fetch for only valid pixels, the quad address.

14. The Output File

The output file is where pixels are put before they go to the RBs. The write BW to this store is 256 bits/clock. Just before this output file are staging registers with write BW 512 bits/clock and read BW 256 bits/clock. The staging registers are 4x128 (and there are 16 of those on the whole chip).

15. IJ Format

The IJ information sent by the PA is of this format on a per quad basis:

We have a vector of IJ's (one IJ per pixel at the centroid of the fragment or at the center of the pixel depending on the mode bit). The interpolation is done at a different precision across the 2x2. The upper left pixel's parameters are always interpolated at full 20x24 mantissa precision. Then the result of the interpolation along with the difference in IJ in reduced precision is used to interpolate the parameter for the other three pixels of the 2x2. Here is how we do it:

Assuming P0 is the interpolated parameter at Pixel 0 having the barycentric coordinates I(0), J(0) and so on for P1,P2 and P3. Also assuming that A is the parameter value at V0 (interpolated with I), B is the parameter value at V1 (interpolated with J) and C is the parameter value at V2 (interpolated with (1-I-J).

$$\Delta 01I = I(1) - I(0)$$
$$\Delta 01J = J(1) - J(0)$$
$$\Delta 02I = I(2) - I(0)$$
$$\Delta 02J = J(2) - J(0)$$
$$\Delta 03I = I(3) - I(0)$$
$$\Delta 03J = J(3) - J(0)$$

P0	P1
P2	P3

$$P0 = C + I(0)*(A-C) + J(0)*(B-C)$$
$$P1 = P0 + \Delta 01I*(A-C) + \Delta 01J*(B-C)$$
$$P2 = P0 + \Delta 02I*(A-C) + \Delta 02J*(B-C)$$
$$P3 = P0 + \Delta 03I*(A-C) + \Delta 03J*(B-C)$$

P0 is computed at 20x24 mantissa precision and P1 to P3 are computed at 8X24 mantissa precision. So far no visual degradation of the image was seen using this scheme.

Multiplies (Full Precision): 2
Multiplies (Reduced precision): 6
Subtracts 19x24 (Parameters): 2

Adds: 8

FORMAT OF P0's IJ : Mantissa 20 Exp 4 for I + Sign
Mantissa 20 Exp 4 for J + Sign

FORMAT of Deltas (x3): Mantissa 8 Exp 4 for I + Sign
Mantissa 8 Exp 4 for J + Sign

Total number of bits : 20*2 + 8*6 + 4*8 + 4*2 = 128

All numbers are kept using the un-normalized floating point convention: if exponent is different than 0 the number is normalized if not, then the number is un-normalized. The maximum range for the IJs (Full precision) is +/- 63 and the range for the Deltas is +/- 127.

15.1 Interpolation of constant attributes

Because of the floating point imprecision, we need to take special provisions if all the interpolated terms are the same or if two of the barycentric coordinates are the same.

We start with the premise that if A = B and B = C and C = A, then P0,1,2,3 = A. Since one or more of the IJ terms may be zero, so we extend this to:

```
if (A=B and B=C and C=A)
   P0,1,2,3 = A;
else if ((I = 0) or (J = 0)) and
          ((J = 0) or (1-I-J = 0)) and
          ((1-J-I = 0) or (I = 0))) {
                if(I != 0) {
                    P0 = A;
                } else if(J != 0) {
                    P0 = B;
                } else {
                    P0 = C;
               }
            //rest of the quad interpolated normally
}
else
{
            normal interpolation
}
```

16. Staging Registers

In order for the reuse of the vertices to be 14, the sequencer will have to re-order the data sent IN ORDER by the VGT for it to be aligned with the parameter cache memory arrangement. Given the following group of vertices sent by the VGT:

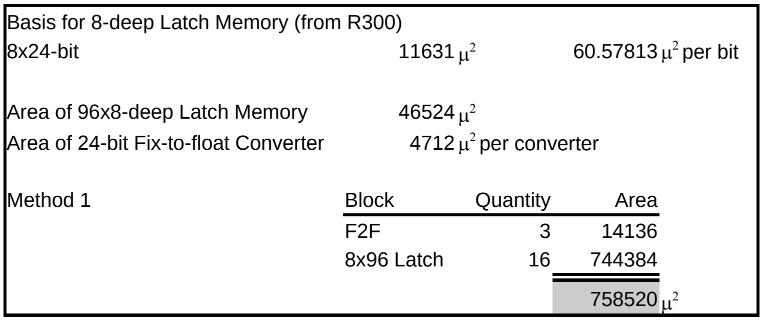
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 || 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 || 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 || 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63

The sequencer will re-arrange them in this fashion:

0 1 2 3 16 17 18 19 32 33 34 35 48 49 50 51 || 4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55 || 8 9 10 11 24 25 26 27 40 41 42 43 56 57 58 59 || 12 13 14 15 28 29 30 31 44 45 46 47 60 61 62 63

The || markers show the SP divisions. In the event a shader pipe is broken, the VGT will send padding to account for the missing pipe. For example, if SP1 is broken, vertices 4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55 will still be sent by the VGT to the SQ **BUT** will not be processed by the SP and thus should be considered invalid (by the SU and VGT).

The most straightforward, *non-compressed* interface method would be to convert, in the VGT, the data to 32-bit floating point prior to transmission to the VSISRs. In this scenario, the data would be transmitted to (and stored in) the VSISRs in full 32-bit floating point. This method requires three 24-bit fixed-to-float converters in the VGT. Unfortunately, it also requires and additional 3,072 bits of storage across the VSISRs. This interface is illustrated in Figure 12~~Figure 12Figure 2~~. The area of the fixed-to-float converters and the VSISRs for this method is roughly estimated as 0.759sqmm using the R300 process. The gate count estimate is shown in Figure 11~~Figure 11Figure 1~~.



Basis for 8-deep Latch Memory (from R300)			
8x24-bit	11631 μ^2		60.57813 μ^2 per bit
Area of 96x8-deep Latch Memory	46524 μ^2		
Area of 24-bit Fix-to-float Converter	4712 μ^2 per converter		
Method 1	Block	Quantity	Area
	F2F	3	14136
	8x96 Latch	16	744384
			758520 μ^2

Figure 11~~111~~:Area Estimate for VGT to Shader Interface

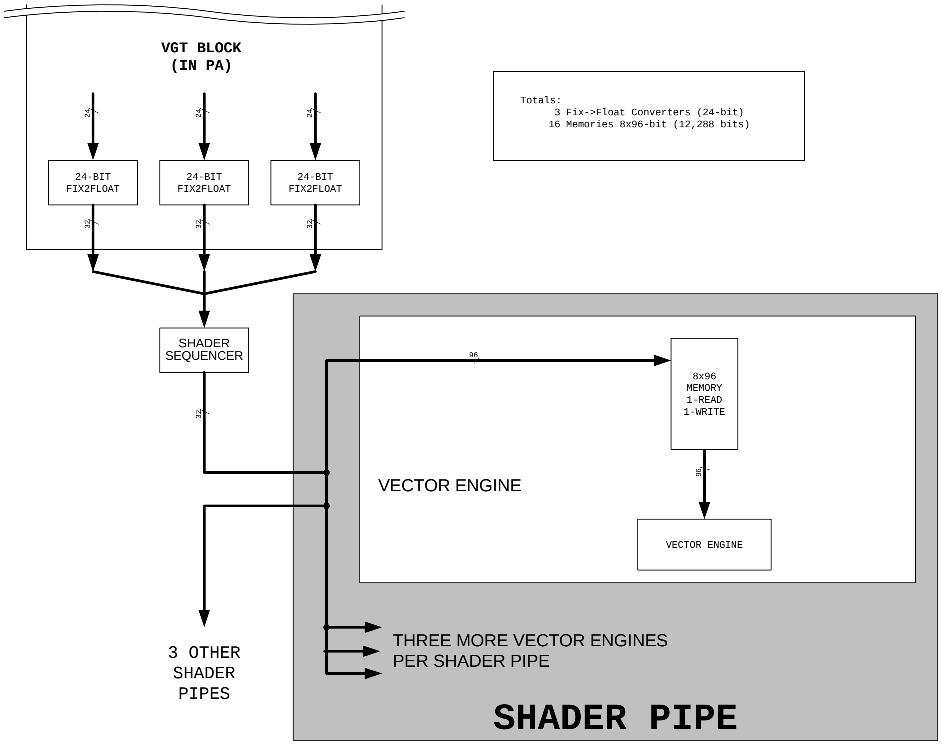


Figure 12~~122~~:VGT to Shader Interface

17. The parameter cache

The parameter cache is where the vertex shaders export their data. It consists of 16 128x128 memories (1R/1W). The reuse engine will make it so that all vertexes of a given primitive will hit different memories. The allocation method for these memories is a simple round robin. The parameter cache pointers are mapped in the following way: 4MSBs are the memory number and the 7 LSBs are the address within this memory.

MEMORY NUMBER 4 bits	ADDRESS 7 bits

The PA generates the parameter cache addresses as the positions come~~s~~ from the SQ. All it needs to do is keep a Current_Location pointer (7 bits only) and as the positions comes increment the memory number. When the memory number field wraps around, the PA increments the Current_Location by VS_EXPORT_COUNT_~~6~~7 (a snooped register from the SQ). As an example, say the memories are all empty to begin with and the vertex shader is exporting 8 parameters per vertex (VS_EXPORT_COUNT_~~6~~ 7 = 8). The first position received is going to have the PC address 00000000000 the second one 00010000000, third one 00100000000 and so on up to 11110000000. Then the next position received (the 17th) is going to have the address 00000001000, the 18th 00010001000, the 19th 00100001000 and so on. The Current_location is NEVER reset BUT on chip resets. The only thing to be careful about is that if the SX doesn't send you a full group of positions (<64) then you need to fill the address space so that the next group starts correctly aligned (for example if you receive only 33 positions then you need to add ~~1~~2*VS_EXPORT_COUNT_~~6~~ 7to Current_Location and reset the memory count to 0 before the next vector begins).

18. Vertex position exporting

On clause 3 the vertex shader can export to the PA both the vertex position and the point sprite. It can also do so at clause 7 if not done at clause 3. The storage needed to perform the position export is at least 64x128 memories for the position and 64x32 memories for the sprite size. It is going to be taken in the pixel output fifo from the SX blocks. The clause where the position export occurs is specified by the EXPORT_LATE register. If turned on, it means that the export is going to occur at ALU clause 7 if unset position export occurs at clause 3.

19. Exporting Arbitration

Here are the rules for co-issuing exporting ALU clauses.

1) Position exports and position exports cannot be co-issued.

Formatted: Bullets and Numbering

All other types of exports can be co-issued as long as there is place in the receiving buffer.

{ISSUE: Do we move the parameter caches to the SX?}
~~Any type of exporting clause can be co-issued. The sequencer will have to make sure back to back memory exports (position/straight memory exports) are interleaved with NOPs as we don't have the bandwidth to service them at full speed.~~

20. Export Types

The export type (or the location where the data should be put) is specified using the destination address field in the ALU instruction. Here is a list of all possible export modes:

20.1 Vertex Shading

0:15	- 16 parameter cache
16:31	- Empty (Reserved?)
32:43	- 12 vertex exports to the frame buffer and index
44:47	- Empty
48:59	- 12 debug export (interpret as normal vertex export)
60	- export addressing mode
61	- Empty
62	- position ~~sprite size export that goes with position export (point_h,point_w,edgeflag,misc)~~
63	- ~~position~~sprite size export that goes with position export (point_h,point_w,edgeflag,misc)

20.2 Pixel Shading

0	- Color for buffer 0 (primary)
1	- Color for buffer 1
2	- Color for buffer 2
3	- Color for buffer 3
4:7	- Empty
8	- Buffer 0 Color/Fog (primary)
9	- Buffer 1 Color/Fog
10	- Buffer 2 Color/Fog
11	- Buffer 3 Color/Fog
12:15	- Empty
16:31	- Empty (Reserved?)
32:43	- 12 exports for multipass pixel shaders.
44:47	- Empty
48:59	- 12 debug exports (interpret as normal pixel export)
60	- export addressing mode
61:62	- Empty
63	- Z for primary buffer (Z exported to 'alpha' component)

21. Special Interpolation modes

21.1 Real time commands

We are unable to use the parameter memory since there is no way for a command stream to write into it. Instead we need to add three 16x128 memories (one for each of three vertices x 16 interpolants). These will be mapped onto the register bus and written by type 0 packets, and output to the the parameter busses (the sequencer and/or PA need to be able to address the reatime parameter memory as well as the regular parameter store. For higher performance we should be able able to view them as two banks of 16 and do double buffering allowing one to be loaded, while the other is rasterized with. Most overlay shaders will need 2 or 4 scalar coordinates, one option might be to restrict the memory to 16x64 or 32x64 allowing only two interpolated scalars per cycle, the only problem I see with this is, if we view support for 16 vector-4 interpolants important (true only if we map Microsoft's high priority stream to the realtime stream), then the PA/sequencer need to support a realtime-specific mode where we need to address 32 vectors of parameters instead of 16. This mode is triggered by the primitive type: REAL TIME. The actual memories are in the in the SX blocks. The parameter data memories are hooked on the RBBM bus and are loaded by the CP using register mapped memory.

21.2 Sprites/ XY screen coordinates/ FB information

When working with sprites, one may want to overwrite the parameter 0 with SC generated data. Also, XY screen coordinates may be needed in the shader program. This functionality is controlled by the gen_I0 register (in SQ) in conjunction with the SND_XY register (in SC). Also it is possible to send the faceness information (for OGL front/back special operations) to the shader using the same control register. Here is a list of all the modes and how they interact together:

Gen_st is a bit taken from the interface between the SC and the SQ. This is the MSB of the primitive type. If the bit is set, it means we are dealing with Point AA, Line AA or sprite and in this case the vertex values are going to generated between 0 and 1.

Param_Gen_I0 disable, snd_xy disable, no gen_st – I0 = No modification
Param_Gen_I0 disable, snd_xy disable, gen_st – I0 = No modification
Param_Gen_I0 disable, snd_xy enable, no gen_st – I0 = No modification
Param_Gen_I0 disable, snd_xy enable, gen_st – I0 = No modification
Param_Gen_I0 enable, snd_xy disable, no gen_st – I0 = garbage, garbage, garbage, faceness
Param_Gen_I0 enable, snd_xy disable, gen_st – I0 = garbage, garbage, s, t
Param_Gen_I0 enable, snd_xy enable, no gen_st – I0 = screen x, screen y, garbage, faceness
Param_Gen_I0 enable, snd_xy enable, gen_st – I0 = screen x, screen y, s, t

21.3 Auto generated counters

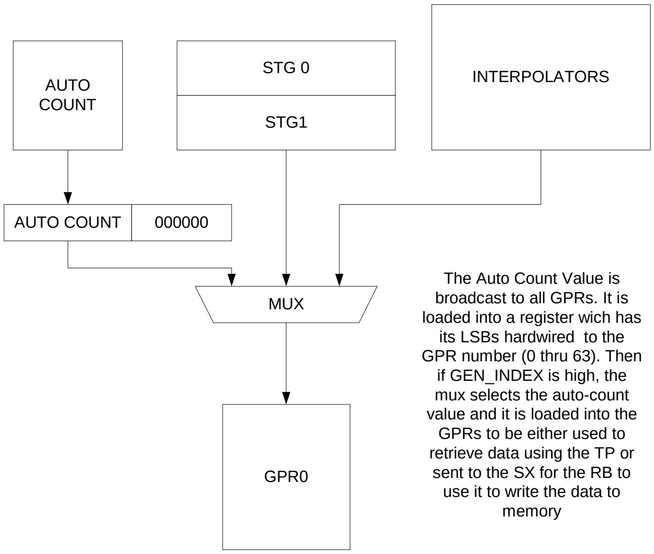
In the cases we are dealing with multipass shaders, the sequencer is going to generate a vector count to be able to both use this count to write the 1^{st} pass data to memory and then use the count to retrieve the data on the 2^{nd} pass. The count is always generated in the same way but it is passed to the shader in a slightly different way depending on the shader type (pixel or vertex). This is toggled on and off using the GEN_INDEX register. The sequencer is going to keep two counters, one for pixels and one for vertices. Every time a full vector of vertices or pixels is written to the GPRs the counter is incremented. Every time a state change is detected, the corresponding counter is reset. While there is only one count broadcast to the GPRs, the LSB are hardwired to specific values making the index different for all elements in the vector.

21.3.1 *Vertex shaders*

In the case of vertex shaders, if GEN_INDEX is set, the data will be put into the x field of the third register (it means that the compiler must allocate 3 GPRs in all multipass vertex shader modes).

21.3.2 *Pixel shaders*

In the case of pixel shaders, if GEN_INDEX is set and Param_Gen_I0 is enabled, the data will be put in the x field of the 2^{nd} register (R1.x), else if GEN_INDEX is set the data will be put into the x field of the 1^{st} register (R0.x).



The Auto Count Value is broadcast to all GPRs. It is loaded into a register wich has its LSBs hardwired to the GPR number (0 thru 63). Then if GEN_INDEX is high, the mux selects the auto-count value and it is loaded into the GPRs to be either used to retrieve data using the TP or sent to the SX for the RB to use it to write the data to memory

Figure 13: GPR input mux Control

22. State management

Every clock, the sequencer will report to the CP the oldest states still in the pipe. These are the states of the programs as they enter the last ALU clause.

22.1 Parameter cache synchronization

In order for the sequencer not to begin a group of pixels before the associated group of vertices has finished, the sequencer will keep a 6 bit count per state (for a total of 8 counters). These counters are initialized to 0 and every time a vertex shader exports its data TO THE PARAMETER CACHE, the corresponding pointer is incremented. When the SC sends a new vector of pixels with the SC_SQ_new_vector bit asserted, the sequencer will first check if the count is greater than 0 before accepting the transmission (it will in fact accept the transmission but then lower its ready to receive). Then the sequencer waits for the count to go to one and decrements it. The sequencer can then issue the group of pixels to the interpolators. Every time the state changes, the new state counter is initialized to 0.

23. XY Address imports

The SC will be able to send the XY addresses to the GPRs. It does so by interleaving the writes of the IJs (to the IJ buffer) with XY writes (to the XY buffer). Then when writing the data to the GPRs, the sequencer is going to interpolate the IJ data or pass the XY data thru a Fix→float converter and expander and write the converted values to the GPRs. The Xys are currently SCREEN SPACE COORDINATES. The values in the XY buffers will wrap. See section 21.2 for details on how to control the interpolation in this mode.

23.1 Vertex indexes imports

In order to import vertex indexes, we have 16 8x96 staging registers. These are loaded one line at a time by the VGT block (96 bits). They are loaded in floating point format and can be transferred in 4 or 8 clocks to the GPRs.

24. Registers

24.1 Control

REG_DYNAMIC	Dynamic allocation (pixel/vertex) of the register file on or off.
REG_SIZE_PIX	Size of the register file's pixel portion (minimal size when dynamic allocation turned on)
REG_SIZE_VTX	Size of the register file's vertex portion (minimal size when dynamic allocation turned on)
ARBITRATION_POLICY	policy of the arbitration between vertexes and pixels
INST_STORE_ALLOC	interleaved, separate
INST_BASE_VTX	start point for the vertex instruction store (RT always ends at vertex_base and Begins at 0)
INST_BASE_PIX	start point for the pixel shader instruction store
ONE_THREAD	debug state register. Only allows one program at a time into the GPRs
ONE_ALU	debug state register. Only allows one ALU program at a time to be executed (instead of 2)
INSTRUCTION~~_ADDR~~	This is where the CP puts the base address of the instruction writes and type (auto-incremented on reads/writes) Register mapped
~~INSTRUCTION_DATA~~	~~This is where the CP puts the actual data going to the instruction memory~~
CONSTANTS	512*4 ALU constants + 32*6 Texture state 32 bits registers (logically mapped)
~~INSTRUCTION_ADDR_RT~~	~~This is where the CP puts the base address of the instruction writes and type for Real Time (auto-incremented on reads/writes)~~
~~INSTRUCTION_DATA_RT~~	~~This is where the CP puts the actual data going to the instruction memory for Real Time~~
CONSTANTS_RT	256*4 ALU constants + 32*6 texture states? (physically mapped)
CONSTANT_EO_RT	This is the size of the space reserved for real time in the constant store (from 0 to CONSTANT_EO_RT). The re-mapping table operates on the rest of the memory
TSTATE_EO_RT	This is the size of the space reserved for real time in the fetch state store (from 0 to TSTATE_EO_RT). The re-mapping table operates on the rest of the memory
EXPORT_LATE	Controls whether or not we are exporting position from clause 3. If set, position exports occur at clause 7.

24.2 Context

VS_FETCH_{0…7}	eight 8 bit pointers to the location where each clauses control program is located
VS_ALU_{0…7}	eight 8 bit pointers to the location where each clauses control program is located
PS_FETCH_{0…7}	eight 8 bit pointers to the location where each clauses control program is located
PS_ALU_{0…7}	eight 8 bit pointers to the location where each clauses control program is located
PS_BASE	base pointer for the pixel shader in the instruction store
VS_BASE	base pointer for the vertex shader in the instruction store
VS_CF_SIZE	size of the vertex shader (# of instructions in control program/2)
PS_CF_SIZE	size of the pixel shader (# of instructions in control program/2)
PS_SIZE	size of the pixel shader (cntl+instructions)
VS_SIZE	size of the vertex shader (cntl+instructions)
PS_NUM_REG	number of GPRs to allocate for pixel shader programs
VS_NUM_REG	number of GPRs to allocate for vertex shader programs
PARAM_SHADE	One 16 bit register specifying which parameters are to be gouraud shaded (0 = flat, 1 = gouraud)
PARAM_WRAP	64 bits: for which parameters (and channels (xyzw)) do we do the cyl wrapping (0=linear, 1=cylindrical).
PS_EXPORT_MODE	0xxxx : Normal mode 1xxxx : Multipass mode If normal, bbbz where bbb is how many colors (0-4) and z is export z or not If multipass 1-12 exports for color.
VS_EXPORT_MASK	which of the last 6 ALU clauses is exporting (multipass only)
VS_EXPORT_MODE	0: position (1 vector), 1: position (2 vectors), 3:multipass
VS_EXPORT	

_COUNT_{0…6}	Six 4 bit counters representing the # of interpolated parameters exported in clause 7 (located in VS_EXPORT_COUNT_6) OR # of exported vectors to memory per clause in multipass mode (per clause)
PARAM_GEN_I0	Do we overwrite or not the parameter 0 with XY data and generated T and S values
GEN_INDEX	Auto generates an address from 0 to XX. Puts the results into R0-1 for pixel shaders and R2 for vertex shaders
CONST_BASE_VTX (9 bits)	Logical Base address for the constants of the Vertex shader
CONST_BASE_PIX (9 bits)	Logical Base address for the constants of the Pixel shader
CONST_SIZE_PIX (8 bits)	Size of the logical constant store for pixel shaders
CONST_SIZE_VTX (8 bits)	Size of the logical constant store for vertex shaders
INST_PRED_OPTIMIZE	Turns on the predicate bit optimization (if of, conditional_execute_predicates is always executed).
CF_BOOLEANS	256 boolean bits
CF_LOOP_COUNT	32x8 bit counters (number of times we traverse the loop)
CF_LOOP_START	32x8 bit counters (init value used in index computation)
CF_LOOP_STEP	32x8 bit counters (step value used in index computation)

25. DEBUG Registers

25.1 Context

DB_PROB_ADDR	instruction address where the first problem occurred
DB_PROB_COUNT	number of problems encountered during the execution of the program
DB_PROB_BREAK	break the clause if an error is found.
DB_INST_COUNT	instruction counter for debug method 2
DB_BREAK_ADDR	break address for method number 2
DB_CLAUSE _MODE_ALU_{0…7}	clause mode for debug method 2 (0: normal, 1: addr, 2: kill)
DB_CLAUSE _MODE_FETCH_{0…7}	clause mode for debug method 2 (0: normal, 1: addr, 2: kill)

25.2 Control

Formatted: Bullets and Numbering

DB_ALUCST_MEMSIZE	Size of the physical ALU constant memory
DB_TSTATE_MEMSIZE	Size of the physical texture state memory

26. Interfaces

26.1 External Interfaces

Whenever an x is used, it means that the bus is broadcast to all units of the same name. For example, if a bus is named SQ→SPx it means that SQ is going to broadcast the same information to all SP instances.

26.1.1 *SC to SQ : IJ Control bus*

This is the control information sent to the sequencer in order to control the IJ fifos and all other information needed to execute a shader program on the sent pixels. This information is sent over 2 clocks, if SENDXY is asserted the next control packet is going to be ignored and XY information is going to be sent on the IJ bus (for the quads that where just sent). All pixels from the group of quads are from the same primitive, all quads of a vector are from the same render state.

Name	Direction	Bits	Description
SC_SQ_q_wr_mask	SC→SQ	4	Quad Write mask left to right
SC_SQ_lod_correct	SC→SQ	24	LOD correction per quad (6 bits per quad)
SC_SQ_flat_vertex	SC→SQ	2	Provoking vertex for flat shading
SC_SQ_param_ptr0	SC→SQ	11	P Store pointer for vertex 0
SC_SQ_param_ptr1	SC→SQ	11	P Store pointer for vertex 1
SC_SQ_param_ptr2	SC→SQ	11	P Store pointer for vertex 2
SC_SQ_end_of_vect	SC→SQ	1	End of the vector
SC_SQ_store_dealloc	SC→SQ	1	Deallocation token for the P Store
SC_SQ_state	SC→SQ	3	State/constant pointer
SC_SQ_valid_pixel	SC→SQ	16	Valid bits for all pixels
SC_SQ_null_prim	SC→SQ	1	Null Primitive (for PC deallocation purposes)
SC_SQ_end_of_prim	SC→SQ	1	End Of the primitive
SC_SQ_send_xy	SC→SQ	1	Sending XY information [XY information is going to be sent on the next clock]
SC_SQ_prim_type	SC→SQ	3	Real time command need to load tex cords from alternate buffer. Line AA, Point AA and Sprite reads their parameters from GEN_T and GEN_S GPRs. 000 : Normal 011 : Real Time 100 : Line AA 101 : Point AA 110 : Sprite
SC_SQ_new_vector	SC→SQ	1	This primitive comes from a new vector of vertices. Make sure that the corresponding vertex shader has finished before starting the group of pixels.
SC_SQ_RTRn	SQ→SC	1	Stalls the PA in n clocks
SC_SQ_RTS	SC→SQ	1	SC ready to send data

26.1.2 *SQ to SP: Interpolator bus*

Name	Direction	Bits	Description
SQ_SPx_interp_prim_type	SQ→SPx	3	Type of the primitive 000 : Normal 011 : Real Time 100 : Line AA 101 : Point AA 110 : Sprite
SQ_SPx_interp_ijline	SQ→SPx	2	Line in the IJ/XY buffer to use to interpolate
SQ_SPx_interp_buff_swap	SQ→SPx	1	Swap the IJ/XY buffers at the end of the interpolation
SQ_SPx_interp_gen_I0	SQ→SPx	1	Generate I0 or not. This tells the interpolators not to use the parameter cache but rather overwrite the data with interpolated 1 and 0. Overwrite if gen_I0 is high.

26.1.3 *SQ to SX: Interpolator bus*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SPx_interp_flat_vtx	SQ→SPx	2	Provoking vertex for flat shading
SQ_SPx_interp_flat_gouraud	SQ→SPx	1	Flat or gouraud shading
SQ_SPx_interp_cyl_wrap	SQ→SPx	4	Wich channel needs to be cylindrical wrapped

~~26.1.3~~26.1.4 *SQ to SP: Parameter Cache Read control bus*

Formatted: Bullets and Numbering

The four following interfaces (SQ→SP, SQ→SX,SP→SX and SX→Interpolators) are all SYNCHRONIZED together.

Name	Direction	Bits	Description
SQ_SPx_ptr0~~SQ_SPx_ptr0~~	SQ→SPx~~SQ→SPx~~	7~~9~~	Parameter Pointer into PC ~~Pointer of PC~~
SQ_SPx_ptr1~~SQ_SPx_ptr1~~	SQ→SPx~~SQ→SPx~~	7~~9~~	Parameter Pointer into PC ~~Pointer of PC~~

SQ_SPx_ptr2~~SQ_SPx_ptr2~~	SQ→SPx~~SQ→SPx~~	7~~9~~	Parameter Pointer into Parameter Cache~~Pointer of PC~~
SQ_SPx_pc0_addr_sel~~SQ_SP0_read_ena~~	SQ→SPx~~SQ→SP0~~	2~~4~~	Selection one of the pointers for parameter cache 0~~Read enables for the 4 memories in the SP0~~
SQ_SPx_pc1_addr_sel~~SQ_SP1_read_ena~~	SQ→SPx~~SQ→SP1~~	2~~4~~	Selection one of the pointers for parameter cache 1~~Read enables for the 4 memories in the SP1~~
SQ_SPx_pc2_addr_sel~~SQ_SP2_read_ena~~	SQ→SPx~~SQ→SP2~~	2~~4~~	Selection one of the pointers for parameter cache 2~~Read enables for the 4 memories in the SP2~~
SQ_SPx_pc3_addr_sel~~SQ_SP3_read_ena~~	SQ→SPx~~SQ→SP3~~	2~~4~~	Selection one of the pointers for parameter cache 3~~Read enables for the 4 memories in the SP3~~
SQ_SP0_read_ena	SQ→SP0	4	Read enables for the 4 memories in the SP0
SQ_SP1_read_ena	SQ→SP1	4	Read enables for the 4 memories in the SP1
SQ_SP2_read_ena	SQ→SP2	4	Read enables for the 4 memories in the SP2
SQ_SP3_read_ena	SQ→SP3	4	Read enables for the 4 memories in the SP3

~~26.1.4~~26.1.5 *SQ to SX: Parameter Cache Mux control Bus*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SXx_mux0	SQ→SXx	4	Mux control for PC or RT (4 MSbs of Pointer in the PC case)
SQ_SXx_mux1	SQ→SXx	4	Mux control for PC or RT (4 MSbs of Pointer in the PC case)
SQ_SXx_mux2	SQ→SXx	4	Mux control for PC or RT (4 MSbs of Pointer in the PC case)
SQ_SXx_RT_switch	SQ→SXx	1	Selects between RT and Normal data

~~26.1.5~~26.1.6 *SQ to SP: Staging Register Data*

Formatted: Bullets and Numbering

This is a broadcast bus that sends the VSISR information to the staging registers of the shader pipes.

Name	Direction	Bits	Description
SQ_SPx_vgt_vsisr_data	SQ→SPx	96	Pointers of indexes or HOS surface information
SQ_SPx_vgt_vsisr_double	SQ→SPx	1	0: Normal 96 bits per vert 1: double 192 bits per vert
SQ_SP0_data_valid	SQ→SP0	1	Data is valid
SQ_SP1_data_valid	SQ→SP1	1	Data is valid
SQ_SP2_data_valid	SQ→SP2	1	Data is valid
SQ_SP3_data_valid	SQ→SP3	1	Data is valid

~~26.1.6~~26.1.7 *PA to SQ : Vertex interface*

Formatted: Bullets and Numbering

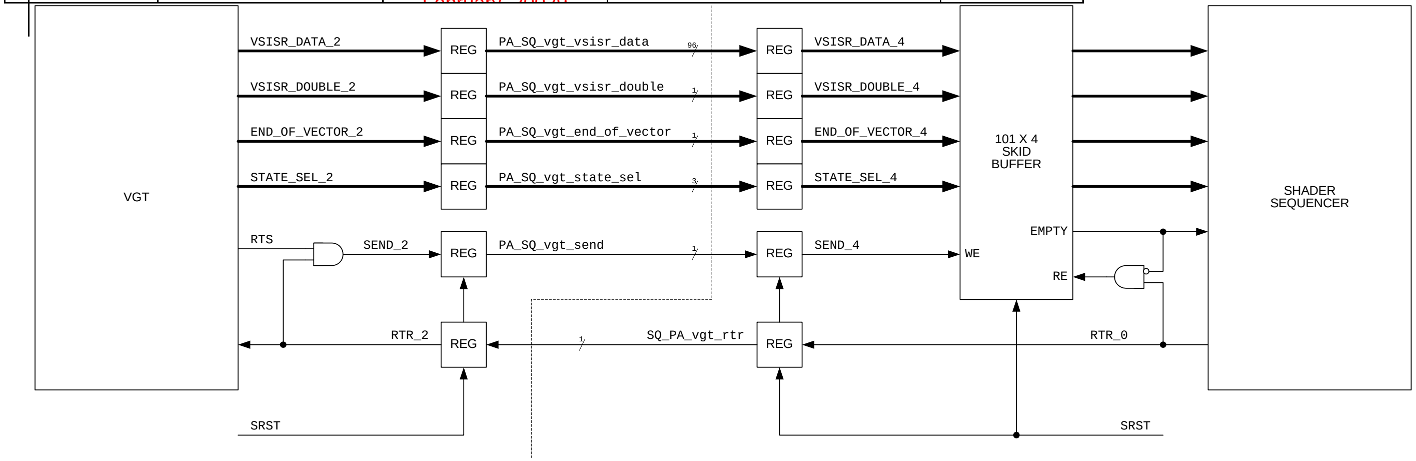
~~26.1.6.1~~26.1.7.1 Interface Signal Table

The area difference between the two methods is not sufficient to warrant complicating the interface or the state requirements of the VSISRs. **Therefore, the POR for this interface is that the VGT will transmit the data to the VSISRs (via the Shader Sequencer) in full, 32-bit floating-point format.** The VGT can transmit up to six 32-bit floating-point values to each VSISR where four or more values require two transmission clocks. The data bus is 96 bits wide.

Name	Bits	Description
PA_SQ_vgt_vsisr_data	~~9~~96~~6~~	Pointers of indexes or HOS surface information
PA_SQ_vgt_vsisr_double	1	0: Normal ~~96~~ 96 bits per vert 1: double ~~192~~ 192 bits per vert
PA_SQ_vgt_end_of_vector	1	Indicates the last VSISR data set for the current process vector (for double vector data, "end_of_vector" is set on the second vector)
PA_SQ_vgt_vsisr_valid	1	Vsisr data is valid
PA_SQ_vgt_state	3	Render State (6*3+3 for constants). This signal is guaranteed to be correct when “PA_SQ_vgt_end_of_vector” is high.
PA_SQ_vgt_send	1	Data on the VGT_SQ is valid receive (see write-up for standard R400 SEND/RTR interface handshaking)
SQ_PA_vgt_rtr	1	Ready to receive (see write-up for standard R400 SEND/RTR interface handshaking)

~~26.1.6.2~~26.1.7.2 Interface Diagrams

Formatted: Bullets and Numbering


VGT
VSISR_DATA_2
VSISR_DOUBLE_2
END_OF_VECTOR_2
STATE_SEL_2
RTS
SEND_2
RTR_2
SRST
REG
PA_SQ_vgt_vsisr_data
96
PA_SQ_vgt_vsisr_double
1
PA_SQ_vgt_end_of_vector
1
PA_SQ_vgt_state_sel
3
PA_SQ_vgt_send
1
SQ_PA_vgt_rtr
1
VSISR_DATA_4
VSISR_DOUBLE_4
END_OF_VECTOR_4
STATE_SEL_4
SEND_4
101 X 4
SKID
BUFFER
EMPTY
WE
RE
RTR_0
SRST
SHADER
SEQUENCER

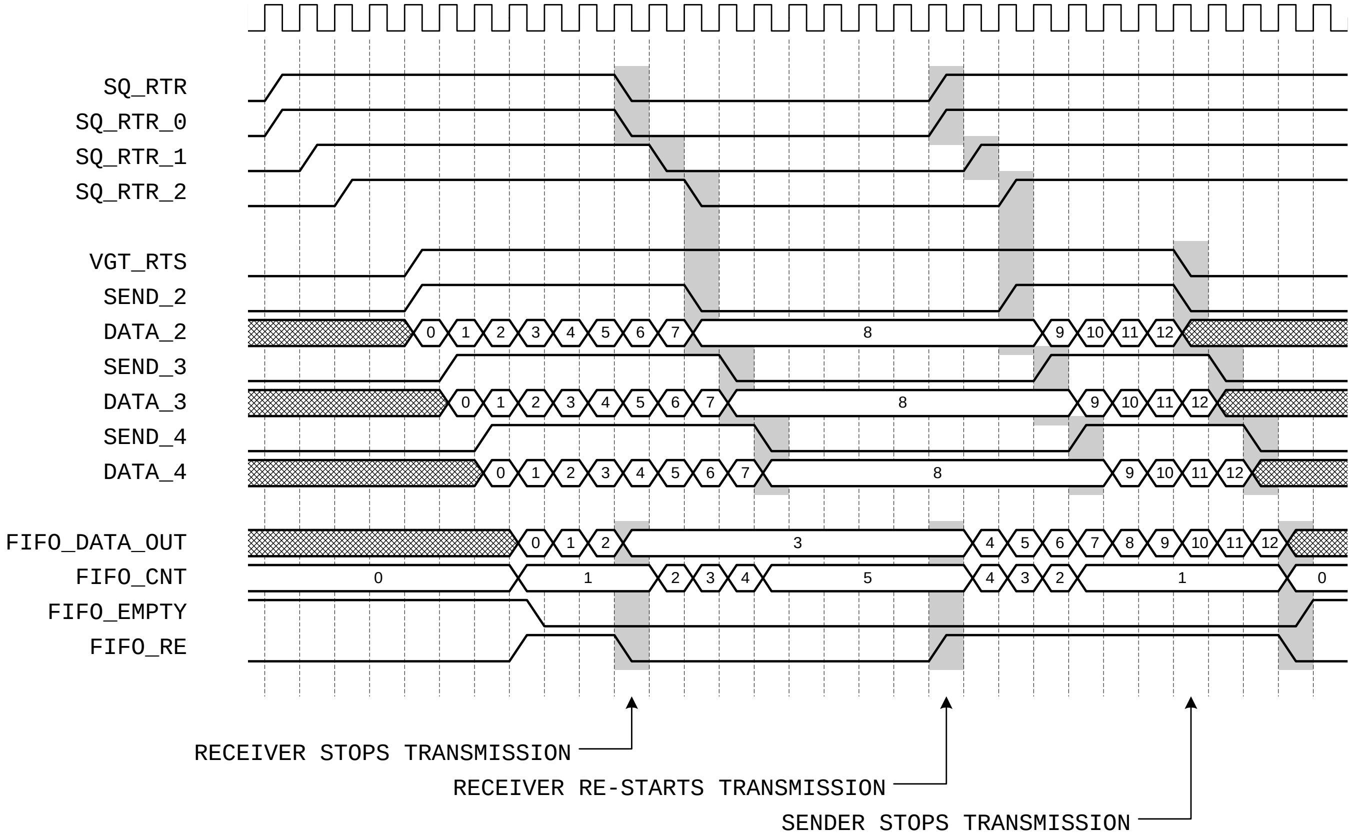


Figure 1. Detailed Logical Diagram for PA_SQ_vgt Interface.

~~26.1.7~~26.1.8 SQ to CP: State report

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_CP_vrtx_state	SEQ→CP	3	Oldest vertex state still in the pipe
SQ_CP_pix_state	SEQ→CP	3	Oldest pixel state still in the pipe

~~26.1.8~~26.1.9 SQ to SX: Control bus

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SXx_exp_Pixel	SQ→SXx	1	1: Pixel 0: Vertex
SQ_SXx_exp_start	SQ→SXx	1	Raised to indicate that the SQ is starting an export
SQ_SXx_exp_Clause	SQ→SXx	3	Clause number, which is needed for vertex clauses
SQ_SXx_exp_State	SQ→SXx	3	State ID, which is needed for vertex clauses

These fields are sent synchronously with SP export data, described in SP0→SX0 interface
{ISSUE: Where are the PC pointers}

~~26.1.9~~26.1.10 SX to SQ : Output file control

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SXx_SQ_Export_count_rdy	SXx→SQ	1	Raised by SX0 to indicate that the following two fields reflect the result of the most recent export
SXx_SQ_Export_Position	SXx→SQ	1	Specifies whether there is room for another position.
SXx_SQ_Export_Buffer	SXx→SQ	7	Specifies the space available in the output buffers. 0: buffers are full 1: 2K-bits available (32-bits for each of the 64 pixels in a clause) ... 64: 128K-bits available (16 128-bit entries for each of 64 pixels) 65-127: RESERVED

~~26.1.10~~26.1.11 SQ to TP: Control bus

Formatted: Bullets and Numbering

Once every clock, the fetch unit sends to the sequencer on which clause it is now working and if the data in the GPRs is ready or not. This way the sequencer can update the fetch counters for the reservation station fifos. The sequencer also provides the instruction and constants for the fetch to execute and the address in the register file where to write the fetch return data.

Name	Direction	Bits	Description
TPx_SQ_data_rdy	TPx→ SQ	1	Data ready
TPx_SQ_clause_num	TPx→ SQ	3	Clause number
TPx_SQ_Type~~TPx_SQ_clause_num~~	TPx→ SQ~~TPx→ SQ~~	1~~3~~	Type of data sent (0:PIXEL, 1:VERTEX)~~Clause number~~
SQ_TPx_const	SQ→TPx	48	Fetch state sent over 4 clocks (192 bits total)
SQ_TPx_instuct	SQ→TPx	24	Fetch instruction sent over 4 clocks
SQ_TPx_end_of_clause	SQ→TPx	1	Last instruction of the clause
SQ_TPx_Type	SQ→TPx	1	Type of data sent (0:PIXEL, 1:VERTEX)
SQ_TPx_phase	SQ→TPx	2	Write phase signal
SQ_TP0_lod_correct	SQ→TP0	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP0_pmask	SQ→TP0	4	Pixel mask 1 bit per pixel
SQ_TP1_lod_correct	SQ→TP1	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP1_pmask	SQ→TP1	4	Pixel mask 1 bit per pixel
SQ_TP2_lod_correct	SQ→TP2	6	LOD correct 3 bits per comp 2 components per quad

SQ_TP2_pmask	SQ→TP2	4	Pixel mask 1 bit per pixel
SQ_TP3_lod_correct	SQ→TP3	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP3_pmask	SQ→TP3	4	Pixel mask 1 bit per pixel
SQ_TPx_clause_num	SQ→TPx	3	Clause number
SQ_TPx_write_gpr_index	SQ->TPx	7	Index into Register file for write of returned Fetch Data

~~26.1.11~~26.1.12 *TP to SQ: Texture stall*

Formatted: Bullets and Numbering

The TP sends this signal to the SQ when its input buffer is full. The SQ is going to send it to the SP X clocks after reception (maximum of 3 clocks of pipeline delay).

Name	Direction	Bits	Description
TP_SQ_fetch_stall	TP→ SQ	1	Do not send more texture request if asserted

~~26.1.12~~26.1.13 *SQ to SP: Texture stall*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SPx_fetch_stall	SQ→SPx	1	Do not send more texture request if asserted

~~26.1.13~~26.1.14 *SQ to SP: GPR, Parameter cache control and auto counter*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SPx_wr_addr	SQ→SPx	7	Write address
SQ_SPx_gpr_rd_addr	SQ→SPx	7	Read address
SQ_SPx_gpr_re_addr	SQ→SPx	1	Read Enable
SQ_SPx_gpr_we_addr	SQ→SPx	1	Write Enable for the GPRs
SQ_SPx_gpr_phase_mux	SQ→SPx	2	The phase mux (arbitrates between inputs, ALU SRC reads and writes)
SQ_SPx_channel_mask	SQ→SPx	4	The channel mask
SQ_SP0_pixel_mask	SQ→SP0	4	The pixel mask
SQ_SP1_pixel_mask	SQ→SP1	4	The pixel mask
SQ_SP2_pixel_mask	SQ→SP2	4	The pixel mask
SQ_SP3_pixel_mask	SQ→SP3	4	The pixel mask
SQ_SPx_pc_we_addr	SQ→SPx	1	Write Enable for the parameter caches
SQ_SPx_gpr_input_mux	SQ→SPx	2	When the phase mux selects the inputs this tells from which source to read from: Interpolated data, VTX0, VTX1, autogen counter.
SQ_SPx_index_count	SQ→SPx	12?	Index count, common for all shader pipes

~~26.1.14~~26.1.15 SQ to SPx: Instructions

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SPx_instruct_start	SQ→SPx	1	Instruction start
SQ_SP_instruct	SQ→SPx	21~~20~~	Transferred over 4 cycles 0: SRC A Select 2:0 SRC A Argument Modifier 3:3 SRC A swizzle 11:4 Unused 20:12 -- - 1: SRC B Select 2:0 SRC B Argument Modifier 3:3 SRC B swizzle 11:4 Unused 20:12 -- - 2: SRC C Select 2:0 SRC C Argument Modifier 3:3 SRC C swizzle 11:4 Unused 20:12 -- - 3: Vector Opcode 4:0 Scalar Opcode 10:5 Vector Clamp 11:11 Scalar Clamp 12:12 Vector Write Mask 16:13 Scalar Write Mask 20:17~~Instruction sent over 4 clocks~~
SQ_SPx_stall	SQ→SPx	1	Stall signal
SQ_SPx_export_count	SQ→SPx	3	Each set of four pixels or vectors is exported over eight clocks. This field specifies where the SP is in that sequence.
SQ_SPx_export_last	SQ→SPx	1	Asserted on the first shader count of the last export of the clause
SQ_SP0_export_pvalid	SQ→SP0	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP0_export_wvalid	SQ→SP0	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors
SQ_SP1_ export_pvalid	SQ→SP1	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP1_ export_wvalid	SQ→SP1	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors
SQ_SP2_ export_pvalid	SQ→SP2	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP2_ export_wvalid	SQ→SP2	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors
SQ_SP3_ export_pvalid	SQ→SP3	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per

			clock
SQ_SP3_ export_wvalid	SQ→SP3	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors

~~26.1.15~~26.1.16 *SP to SQ: Constant address load/ Predicate Set*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SP0_SQ_const_addr	SP0→SQ	36	Constant address load / predicate vector load (4 bits only) to the sequencer
SP0_SQ_valid	SP0→SQ	1	Data valid
SP1_SQ_const_addr	SP1→SQ	36	Constant address load / predicate vector load (4 bits only) to the sequencer
SP1_SQ_valid	SP1→SQ	1	Data valid
SP2_SQ_const_addr	SP2→SQ	36	Constant address load / predicate vector load (4 bits only) to the sequencer
SP2_SQ_valid	SP2→SQ	1	Data valid
SP3_SQ_const_addr	SP3→SQ	36	Constant address load / predicate vector load (4 bits only) to the sequencer
SP3_SQ_valid	SP3→SQ	1	Data valid

~~26.1.16~~26.1.17 *SQ to SPx: constant broadcast*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SPx_constant	SQ→SPx	128	Constant broadcast

~~26.1.17~~26.1.18 *SP0 to SQ: Kill vector load*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SP0_SQ_kill_vect	SP0→SQ	4	Kill vector load
SP1_SQ_kill_vect	SP1→SQ	4	Kill vector load
SP2_SQ_kill_vect	SP2→SQ	4	Kill vector load
SP3_SQ_kill_vect	SP3→SQ	4	Kill vector load

~~26.1.18~~26.1.19 *SQ to CP: RBBM bus*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_RBB_rs	SQ→CP	1	Read Strobe
SQ_RBB_rd	SQ→CP	32	Read Data
SQ_RBBM_nrtrtr	SQ→CP	1	Optional
SQ_RBBM_rtr	SQ→CP	1	Real-Time (Optional)

~~26.1.19~~26.1.20 *CP to SQ: RBBM bus*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
rbbm_we	CP→SQ	1	Write Enable
rbbm_a	CP→SQ	~~18~~15	Address -- Upper Extent is TBD (16:2)
rbbm_wd	CP→SQ	32	Data
rbbm_be	CP→SQ	4	Byte Enables
rbbm_re	CP→SQ	1	Read Enable
rbb_rs0	CP→SQ	1	Read Return Strobe 0
rbb_rs1	CP→SQ	1	Read Return Strobe 1
rbb_rd0	CP→SQ	32	Read Data 0
rbb_rd1	CP→SQ	32	Read Data 0
RBBM_SQ_soft_reset	CP→SQ	1	Soft Reset

27. Examples of program executions

27.1.1 *Sequencer Control of a Vector of Vertices*

1. PA sends a vector of 64 vertices (actually vertex indices – 32 bits/index for 2048 bit total) to the RE's Vertex FIFO
    - state pointer as well as tag into position cache is sent along with vertices
    - space was allocated in the position cache for transformed position before the vector was sent
    - **also before the vector is sent to the RE, the CP has loaded the global instruction store with the vertex shader program (using the MH?)**
    - The vertex program is assumed to be loaded when we receive the vertex vector.
        - the SEQ then accesses the IS base for this shader using the local state pointer (provided to all sequencers by the RBBM when the CP is done loading the program)
2. SEQ arbitrates between the Pixel FIFO and the Vertex FIFO – basically the Vertex FIFO always has priority
    - at this point the vector is removed from the Vertex FIFO
    - the arbiter is not going to select a vector to be transformed if the parameter cache is full unless the pipe as nothing else to do (ie no pixels are in the pixel fifo).
3. SEQ allocates space in the SP register file for index data plus GPRs used by the program
    - the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer that came down with the vertices
    - SEQ will not send vertex data until space in the register file has been allocated
4. SEQ sends the vector to the SP register file over the RE_SP interface (which has a bandwidth of 2048 bits/cycle)
    - the 64 vertex indices are sent to the 64 register files over 4 cycles
        - RF0 of SU0, SU1, SU2, and SU3 is written the first cycle
        - RF1 of SU0, SU1, SU2, and SU3 is written the second cycle
        - RF2 of SU0, SU1, SU2, and SU3 is written the third cycle
        - RF3 of SU0, SU1, SU2, and SU3 is written the fourth cycle
    - the index is written to the least significant 32 bits **(floating point format?) (what about compound indices)** of the 128-bit location within the register file (w); the remaining data bits are set to zero (x, y, z)
5. SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of fetch state machine 0, or TSM0 FIFO)
    - the control packet contains the state pointer, the tag to the position cache and a register file base pointer.
6. TSM0 accepts the control packet and fetches the instructions for fetch clause 0 from the global instruction store
    - TSM0 was first selected by the TSM arbiter before it could start
7. all instructions of fetch clause 0 are issued by TSM0
8. the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)
    - TSM0 does not wait for requests made to the Fetch Unit to complete; it passes the register file write index for the fetch data to the TU, which will write the data to the RF as it is received
    - once the TU has written all the data to the register files, it increments a counter that is associated with ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go ahead start to execute the ALU clause
9. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store
10. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of fetch state machine 1, or TSM1 FIFO)
11. the control packet continues to travel down the path of reservation stations until all clauses have been executed
    - position can be exported in ALU clause 3 (or 4?); the data (and the tag) is sent over a position bus (which is shared with all four shader pipes) back to the PA's position cache
    - A parameter cache pointer is also sent along with the position data. This tells to the PA where the data is going to be in the parameter cache.
        - there is a position export FIFO in the SP that buffers position data before it gets sent back to the PA

    - the ASM arbiter will prevent a packet from starting an exporting clause if the position export FIFO is full
- parameter data is exported in clause 7 (as well as position data if it was not exported earlier)
    - parameter data is sent to the Parameter Cache over a dedicated bus
    - the SEQ allocates storage in the Parameter Cache, and the SEQ deallocates that space when there is no longer a need for the parameters (it is told by the PA when using a token).
    - the ASM arbiter will prevent a packet from starting on ASM7 if the parameter cache (or the position buffer if position is being exported) is full

12. after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program

27.1.2 *Sequencer Control of a Vector of Pixels*

1. **As with vertex shader programs, pixel shaders are loaded into the global instruction store by the CP**
    - At this point it is assumed that the pixel program is loaded into the instruction store and thus ready to be read.
2. the RE's Pixel FIFO is loaded with the barycentric coordinates for pixel quads by the detailed walker
    - the state pointer and the LOD correction bits are also placed in the Pixel FIF0
    - the Pixel FIFO is wide enough to source four quad's worth of barycentrics per cycle
3. SEQ arbitrates between Pixel FIFO and Vertex FIFO – when there are no vertices pending OR there is no space left in the register files for vertices, the Pixel FIFO is selected
4. SEQ allocates space in the SP register file for all the GPRs used by the program
    - the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer
    - SEQ will not allow interpolated data to be sent to the shader until space in the register file has been allocated
5. SEQ controls the transfer of interpolated data to the SP register file over the RE_SP interface (which has a bandwidth of 2048 bits/cycle). See interpolated data bus diagrams for details.
6. SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of fetch state machine 0, or TSM0 FIFO)
    - note that there is a separate set of reservation stations/arbiters/state machines for vertices and for pixels
    - the control packet contains the state pointer, the register file base pointer, and the LOD correction bits
    - all other information (such as quad address for example) travels in a separate FIFO
7. TSM0 accepts the control packet and fetches the instructions for fetch clause 0 from the global instruction store
    - TSM0 was first selected by the TSM arbiter before it could start
8. all instructions of fetch clause 0 are issued by TSM0
9. the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)
    - TSM0 does not wait for fetch requests made to the Fetch Unit to complete; it passes the register file write index for the fetch data to the TU, which will write the data to the RF as it is received
    - once the TU has written all the data for a particular clause to the register files, it increments a counter that is associated with the ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go ahead and pop the FIFO and start to execute the ALU clause
10. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store
11. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of fetch state machine 1, or TSM1 FIFO)
12. the control packet continues to travel down the path of reservation stations until all clauses have been executed
    - pixel data is exported in the last ALU clause (clause 7)
        - it is sent to an output FIFO where it will be picked up by the render backend
        - the ASM arbiter will prevent a packet from starting on ASM7 if the output FIFO is full
13. after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program

27.1.3 *Notes*

14. The state machines and arbiters will operate ahead of time so that they will be able to immediately start the real threads or stall.
15. The register file base pointer for a vector needs to travel with the vector through the reservation stations, but the instruction store base pointer does not – this is because the RF pointer is different for all threads, but the IS pointer is only different for each state and thus can be accessed via the state pointer.

28. Open issues

Need to do some testing on the size of the register file as well as on the register file allocation method (dynamic VS static).

Saving power?

Parameter caches in SX?

Using both IJ buffers for center + centroid interpolation?

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 20154 March, 20024 February	GEN-CXXXXX-REVA	1 of 48

Author: Laurent Lefebvre

Issue To:	**Copy No:**

R400 Sequencer Specification

SQ

Version 1.8~~7~~

Overview: This is an architectural specification for the R400 Sequencer block (SEQ). It provides an overview of the required capabilities and expected uses of the block. It also describes the block interfaces, internal sub-blocks, and provides internal state diagrams.

AUTOMATICALLY UPDATED FIELDS:
Document Location: C:\perforce\r400\doc_lib\design\blocks\sq\R400_Sequencer.doc
Current Intranet Search Title: R400 Sequencer Specification

APPROVALS	
Name/Dept	Signature/Date

Remarks:

THIS DOCUMENT CONTAINS CONFIDENTIAL INFORMATION THAT COULD BE SUBSTANTIALLY DETRIMENTAL TO THE INTEREST OF ATI TECHNOLOGIES INC. THROUGH UNAUTHORIZED USE OR DISCLOSURE.

Table Of Contents

1. OVERVIEW 6
1.1 Top Level Block Diagram 8
1.2 Data Flow graph (SP) 10
1.3 Control Graph 11
2. INTERPOLATED DATA BUS 11
3. INSTRUCTION STORE 14
4. SEQUENCER INSTRUCTIONS 16
5. CONSTANT STORES 16
5.1 Memory organizations 16
5.2 Management of the re-mapping tables 16
5.2.1 Dirty bits .... 19~~18~~
5.2.2 Free List Block .... 19~~18~~
5.2.3 De-allocate Block .... 20~~1918~~
5.2.4 Operation of Incremental model .... 20~~1918~~
5.3 Constant Store Indexing .... 20~~19~~
5.4 Real Time Commands .... 21~~2019~~
5.5 Constant Waterfalling .... 21~~2019~~
6. LOOPING AND BRANCHES .... 22~~2120~~
6.1 The controlling state. .... 22~~2120~~
6.2 The Control Flow Program .... 22~~2120~~
6.3 Data dependant predicate instructions .... 24~~23~~
6.4 HW Detection of PV,PS .... 25~~2423~~
6.5 Register file indexing .... 25~~2423~~
6.6 Predicated Instruction support for Texture clauses .... 26~~24~~
6.7 Debugging the Shaders .... 26~~2524~~
6.7.1 Method 1: Debugging registers .... 26~~2524~~
6.7.2 Method 2: Exporting the values in the GPRs (12) .... 26~~25~~
7. PIXEL KILL MASK .... 27~~25~~
8. MULTIPASS VERTEX SHADERS (HOS) .... 27~~2625~~
9. REGISTER FILE ALLOCATION .... 27~~2625~~
10. FETCH ARBITRATION .... 28~~2726~~
11. ALU ARBITRATION .... 28~~2726~~
12. HANDLING STALLS .... 29~~2827~~
13. CONTENT OF THE RESERVATION STATION FIFOS .... 29~~2827~~
14. THE OUTPUT FILE .... 29~~2827~~
15. IJ FORMAT .... 29~~2827~~
15.1 Interpolation of constant attributes .... 30~~2928~~
16. STAGING REGISTERS .... 30~~2928~~
17. THE PARAMETER CACHE .... 32~~3130~~
18. VERTEX POSITION EXPORTING .... 33~~3130~~
19. EXPORTING ARBITRATION .... 33~~3130~~
20. EXPORT TYPES .... 33~~3130~~
20.1 Vertex Shading .... 33~~3130~~

20.2 Pixel Shading .... 33~~3230~~

21. SPECIAL INTERPOLATION MODES .... 34~~3231~~

21.1 Real time commands .... 34~~3231~~

21.2 Sprites/ XY screen coordinates/ FB information .... 34~~3231~~

21.3 Auto generated counters .... 34~~3331~~

21.3.1 Vertex shaders .... 34~~3332~~

21.3.2 Pixel shaders .... 34~~3332~~

22. STATE MANAGEMENT .... 35~~3332~~

22.1 Parameter cache synchronization .... 35~~3332~~

23. XY ADDRESS IMPORTS .... 35~~3432~~

23.1 Vertex indexes imports .... 35~~3433~~

24. REGISTERS .... 36~~3433~~

24.1 Control .... 36~~3433~~

24.2 Context .... 36~~3433~~

25. DEBUG REGISTERS .... 37~~3534~~

25.1 Context .... 37~~3534~~

26. INTERFACES .... 37~~3534~~

26.1 External Interfaces .... 37~~3534~~

26.1.1 SC to SQ : IJ Control bus .... 37~~3634~~

26.1.2 SQ to SP: Interpolator bus .... 38~~3635~~

26.1.3 SQ to SP: Parameter Cache Read control bus .... 38~~3635~~

26.1.4 SQ to SX: Parameter Cache Mux control Bus .... 39~~3736~~

26.1.5 SQ to SP: Staging Register Data .... 39~~3736~~

26.1.6 PA to SQ : Vertex interface .... 39~~3736~~

26.1.7 SQ to CP: State report .... 42~~4139~~

26.1.8 SQ to SX: Control bus .... 42~~4139~~

26.1.9 SX to SQ : Output file control .... 42~~4139~~

26.1.10 SQ to TP: Control bus .... 42~~4139~~

26.1.11 TP to SQ: Texture stall .... 43~~4240~~

26.1.12 SQ to SP: Texture stall .... 43~~4240~~

26.1.13 SQ to SP: GPR, Parameter cache control and auto counter .... 43~~4240~~

26.1.14 SQ to SPx: Instructions .... 44~~4341~~

26.1.15 SP to SQ: Constant address load .... 45~~4441~~

26.1.16 SQ to SPx: constant broadcast .... 45~~4441~~

26.1.17 SP0 to SQ: Kill vector load .... 45~~4442~~

26.1.18 SQ to CP: RBBM bus .... 45~~4442~~

26.1.19 CP to SQ: RBBM bus .... 45~~4442~~

27. EXAMPLES OF PROGRAM EXECUTIONS .... 46~~4442~~

27.1.1 Sequencer Control of a Vector of Vertices .... 46~~4442~~

27.1.2 Sequencer Control of a Vector of Pixels .... 47~~4643~~

27.1.3 Notes .... 48~~4644~~

28. OPEN ISSUES..48~~4744~~

Revision Changes:

Rev 0.1 (Laurent Lefebvre) Date: May 7, 2001	First draft.
Rev 0.2 (Laurent Lefebvre) Date : July 9, 2001	Changed the interfaces to reflect the changes in the SP. Added some details in the arbitration section.
Rev 0.3 (Laurent Lefebvre) Date : August 6, 2001	Reviewed the Sequencer spec after the meeting on August 3, 2001.
Rev 0.4 (Laurent Lefebvre) Date : August 24, 2001	Added the dynamic allocation method for register file and an example (written in part by Vic) of the flow of pixels/vertices in the sequencer.
Rev 0.5 (Laurent Lefebvre) Date : September 7, 2001	Added timing diagrams (Vic)
Rev 0.6 (Laurent Lefebvre) Date : September 24, 2001	Changed the spec to reflect the new R400 architecture. Added interfaces.
Rev 0.7 (Laurent Lefebvre) Date : October 5, 2001	Added constant store management, instruction store management, control flow management and data dependant predication.
Rev 0.8 (Laurent Lefebvre) Date : October 8, 2001	Changed the control flow method to be more flexible. Also updated the external interfaces.
Rev 0.9 (Laurent Lefebvre) Date : October 17, 2001	Incorporated changes made in the 10/18/01 control flow meeting. Added a NOP instruction, removed the conditional_execute_or_jump. Added debug registers.
Rev 1.0 (Laurent Lefebvre) Date : October 19, 2001	Refined interfaces to RB. Added state registers.
Rev 1.1 (Laurent Lefebvre) Date : October 26, 2001	Added SEQ→SP0 interfaces. Changed delta precision. Changed VGT→SP0 interface. Debug Methods added.
Rev 1.2 (Laurent Lefebvre) Date : November 16, 2001	Interfaces greatly refined. Cleaned up the spec.
Rev 1.3 (Laurent Lefebvre) Date : November 26, 2001	Added the different interpolation modes.
Rev 1.4 (Laurent Lefebvre) Date : December 6, 2001	Added the auto incrementing counters. Changed the VGT→SQ interface. Added content on constant management. Updated GPRs.
Rev 1.5 (Laurent Lefebvre) Date : December 11, 2001	Removed from the spec all interfaces that weren't directly tied to the SQ. Added explanations on constant management. Added PA→SQ synchronization fields and explanation.
Rev 1.6 (Laurent Lefebvre) Date : January 7, 2002	Added more details on the staging register. Added detail about the parameter caches. Changed the call instruction to a Conditionnal_call instruction. Added details on constant management and updated the diagram.
Rev 1.7 (Laurent Lefebvre) Date : February 4, 2002	Added Real Time parameter control in the SX interface. Updated the control flow section.
Rev 1.8 (Laurent Lefebvre) Date : March 4, 2002	New interfaces to the SX block. Added the end of clause modifier, removed the end of clause instructions.

1. Overview

The sequencer is based on the R300 design. It chooses two ALU clauses and a fetch clause to execute, and executes all of the instructions in a clause before looking for a new clause of the same type. Two ALU clauses are executed interleaved to hide the ALU latency. Each vector will have eight fetch and eight ALU clauses, but clauses do not need to contain instructions. A vector of pixels or vertices ping-pongs along the sequencer FIFO, bouncing from fetch reservation station to alu reservation station. A FIFO exists between each reservation stage, holding up vectors until the vector currently occupying a reservation station has left. A vector at a reservation station can be chosen to execute. The sequencer looks at all eight alu reservation stations to choose an alu clause to execute and all eight fetch stations to choose a fetch clause to execute. The arbitrator will give priority to clauses/reservation stations closer to the bottom of the pipeline. It will not execute an alu clause until the fetch fetches initiated by the previous fetch clause have completed. There are two separate sets of reservation stations, one for pixel vectors and one for vertices vectors. This way a pixel can pass a vertex and a vertex can pass a pixel.

To support the shader pipe the sequencer also contains the shader instruction cache, constant store, control flow constants and texture state. The four shader pipes also execute the same instruction thus there is only one sequencer for the whole chip.

The sequencer first arbitrates between vectors of 64 vertices that arrive directly from primitive assembly and vectors of 16 quads (64 pixels) that are generated in the scan converter.

The vertex or pixel program specifies how many GPRs it needs to execute. The sequencer will not start the next vector until the needed space is available in the GPRs.

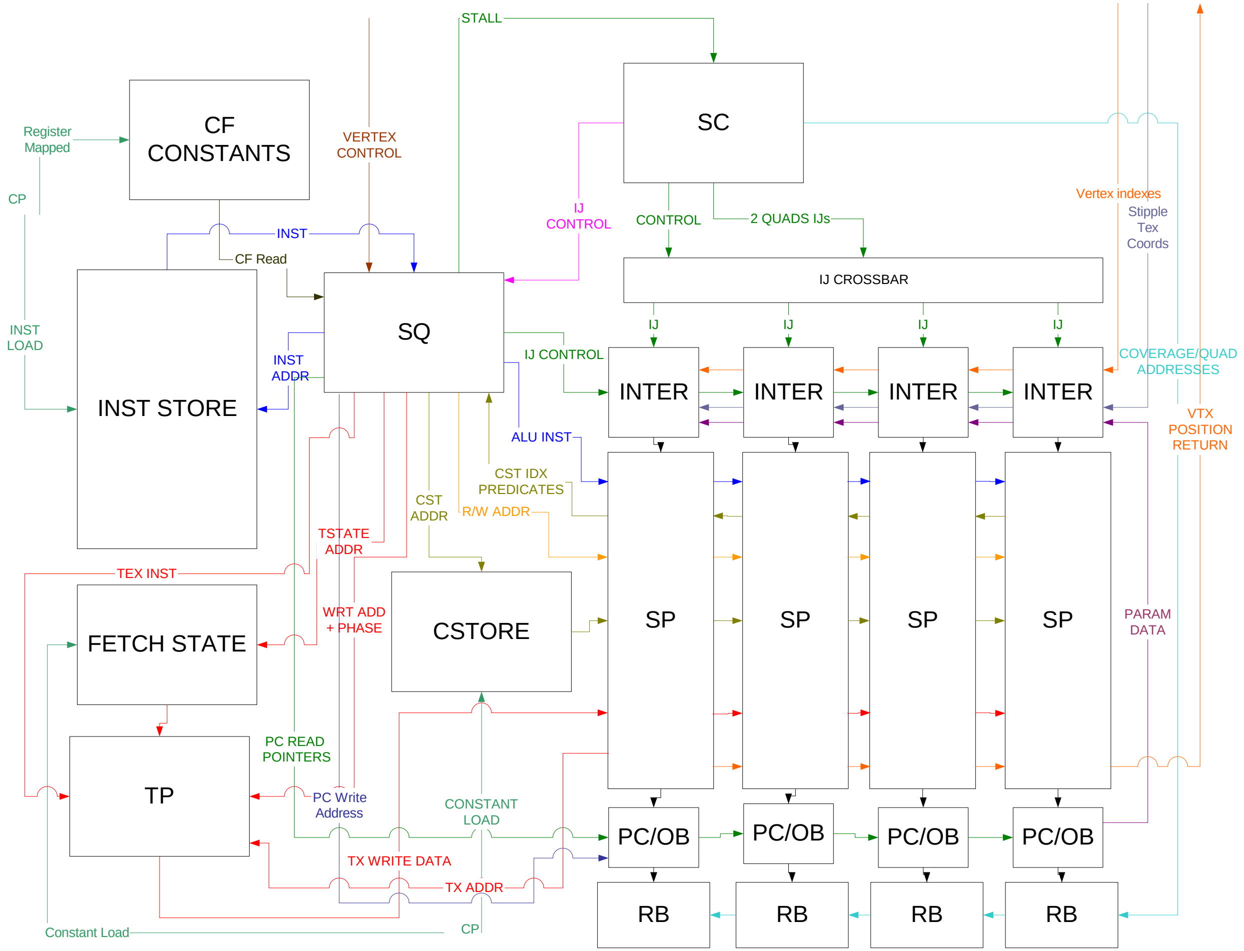
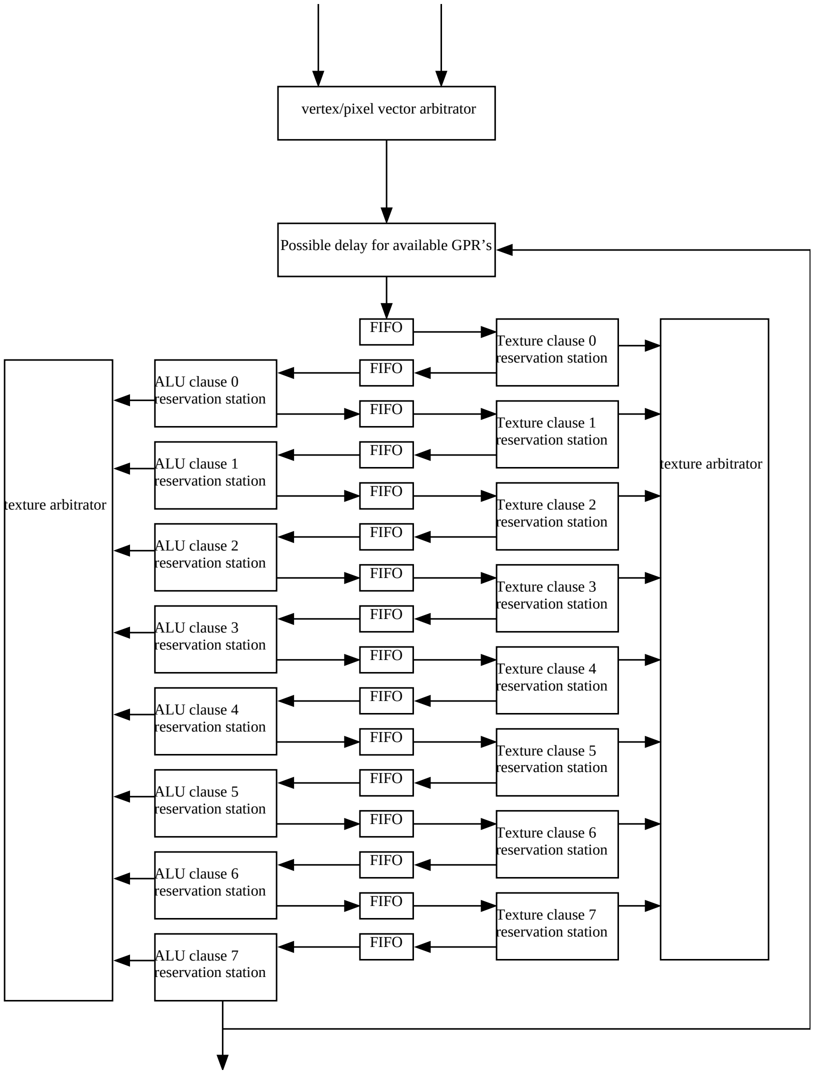


Figure 1: General Sequencer overview

1.1 Top Level Block Diagram



vertex/pixel vector arbitrator
Possible delay for available GPR's
FIFO
Texture clause 0 reservation station
ALU clause 0 reservation station
Texture clause 1 reservation station
ALU clause 1 reservation station
Texture clause 2 reservation station
ALU clause 2 reservation station
Texture clause 3 reservation station
ALU clause 3 reservation station
Texture clause 4 reservation station
ALU clause 4 reservation station
Texture clause 5 reservation station
ALU clause 5 reservation station
Texture clause 6 reservation station
ALU clause 6 reservation station
Texture clause 7 reservation station
ALU clause 7 reservation station
texture arbitrator
texture arbitrator

Figure 2: Reservation stations and arbiters

There are two sets of the above figure, one for vertices and one for pixels.

Depending on the arbitration state, the sequencer will either choose a vertex or a pixel packet. The control packet consists of 3 bits of state, 7 bits for the base address of the Shader program and some information on the coverage to determine fetch LOD plus other various small state bits.

On receipt of a packet, the input state machine (not pictured but just before the first FIFO) allocated enough space in the GPRs to store the interpolated values and temporaries. Following this, the barycentric coordinates (and XY screen position if needed) are sent to the interpolator, which will use them to interpolate the parameters and place the results into the GPRs. Then, the input state machine stacks the packet in the first FIFO.

On receipt of a command, the level 0 fetch machine issues a fetch request to the TP and corresponding GPR address for the fetch address (ta). A small command (tcmd) is passed to the fetch system identifying the current level number (0) as well as the GPR write address for the fetch return data. One fetch request is sent every 4 clocks causing the texturing of sixteen 2x2s worth of data (or 64 vertices). Once all the requests are sent the packet is put in FIFO 1.

Upon receipt of the return data, the fetch unit writes the data to the register file using the write address that was provided by the level 0 fetch machine and sends the clause number (0) to the level 0 fetch state machine to signify that the write is done and thus the data is ready. Then, the level 0 fetch machine increments the counter of FIFO 1 to signify to the ALU 0 that the data is ready to be processed.

On receipt of a command, the level 0 ALU machine first decrements the input FIFO 1 counter and then issues a complete set of level 0 shader instructions. For each instruction, the ALU state machine generates 3 source addresses, one destination address and an instruction. Once the last instruction has been issued, the packet is put into FIFO 2.

There will always be two active ALU clauses at any given time (and two arbiters). One arbiter will arbitrate over the odd instructions (4 clocks cycles) and the other one will arbitrate over the even instructions (4 clocks cycles). The only constraints between the two arbiters is that they are not allowed to pick the same clause number as the other one is currently working on if the packet is not of the same type (render state).

If the packet is a vertex packet, upon reaching ALU clause 3, it can export the position if the position is ready. So the arbiter must prevent ALU clause 3 to be selected if the positional buffer is full (or can't be accessed). Along with the positional data, if needed the sprite size and/or edge flags can also be sent.

A special case is for multipass vertex shaders, which can export 12 parameters per last 6 clauses to the output buffer. If the output buffer is full or doesn't have enough space the sequencer will prevent such a vertex group to enter an exporting clause.

Multipass pixel shaders can export 12 parameters to memory from the last clause only (7).

All other clauses process in the same way until the packet finally reaches the last ALU machine (7).

Only one pair of interleaved ALU state machines may have access to the register file address bus or the instruction decode bus at one time. Similarly, only one fetch state machine may have access to the register file address bus at one time. Arbitration is performed by three arbiter blocks (two for the ALU state machines and one for the fetch state machines). The arbiters always favor the higher number state machines, preventing a bunch of half finished jobs from clogging up the register files.

1.2 Data Flow graph (SP)

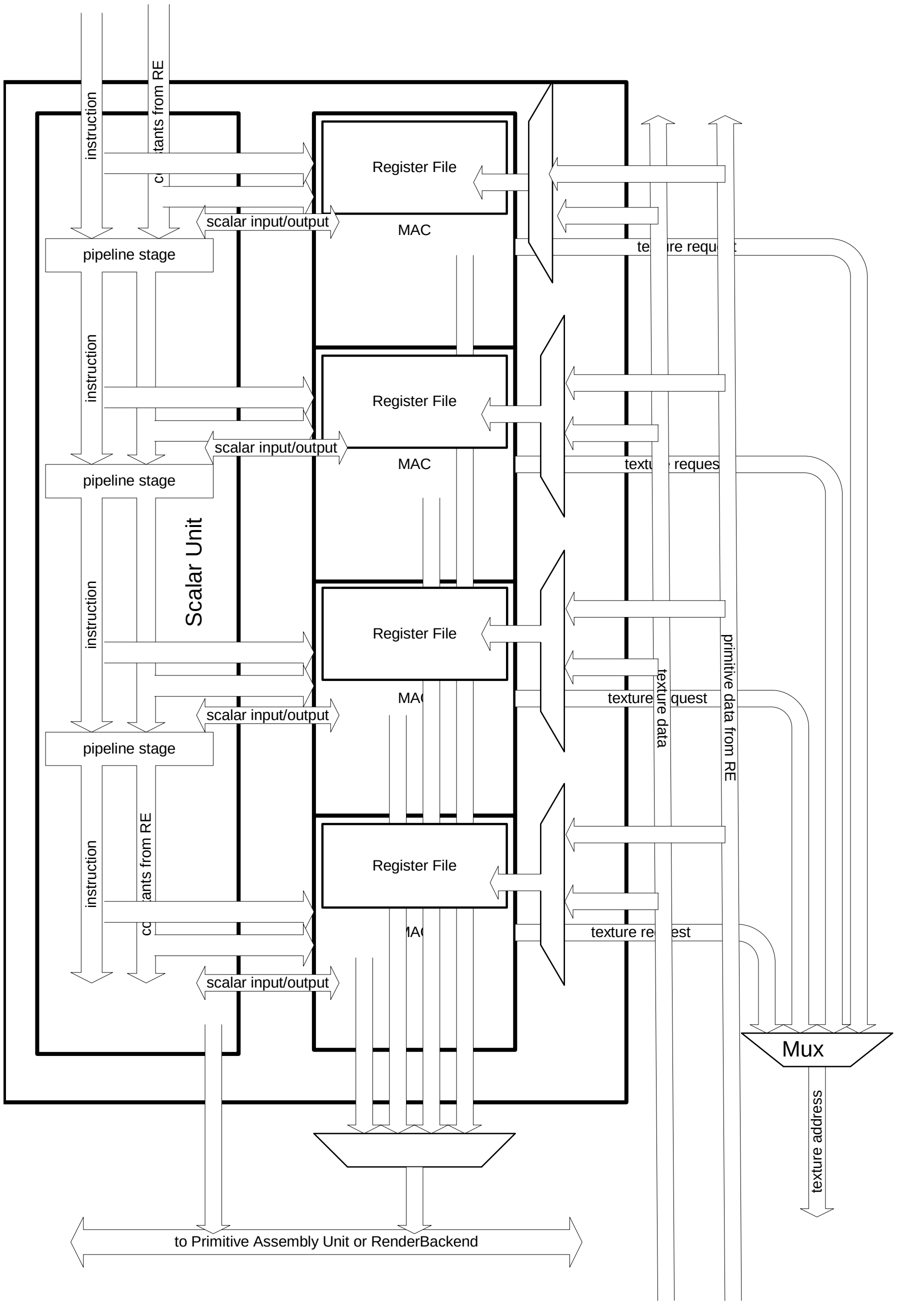


Figure 3: The shader Pipe

The gray area represents blocks that are replicated 4 times per shader pipe (16 times on the overall chip).

1.3 Control Graph

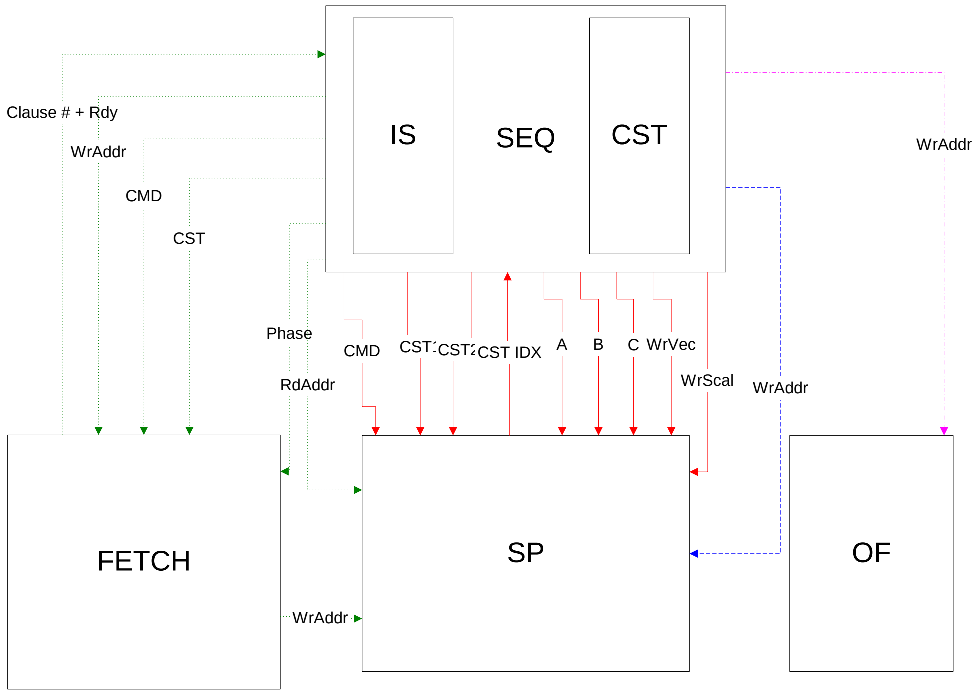


Figure 4: Sequencer Control interfaces

In green is represented the Fetch control interface, in red the ALU control interface, in blue the Interpolated/Vector control interface and in purple is the output file control interface.

2. Interpolated data bus

The interpolators contain an IJ buffer to pack the information as much as possible before writing it to the register file.

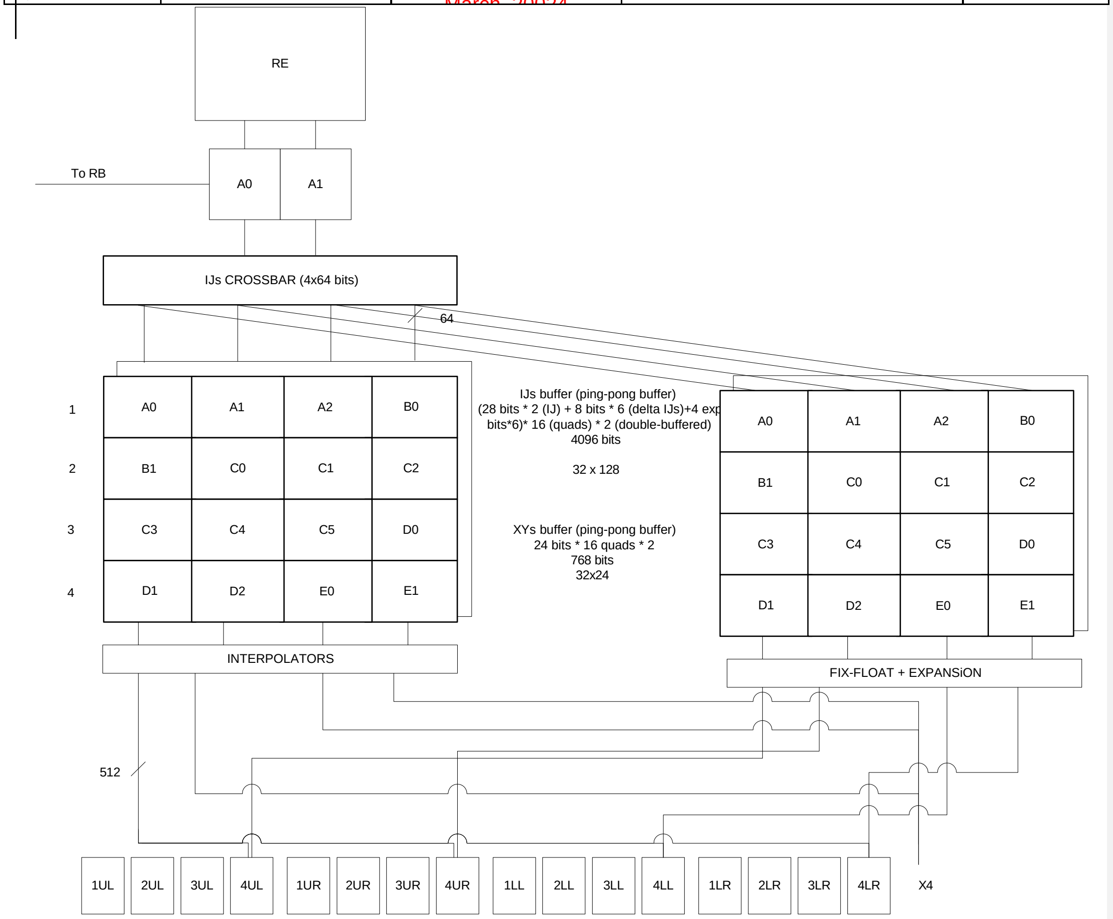


Figure 5: Interpolation buffers

WRITES

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21	T22	T23
SP 0	A0	A0	XY A0	B1	B1	XY B1	C3	C3	XY C3				D1	D1	XY D1									
SP 1	A1	A1	XY A1				C0	C0	XY C0	C4	C4	XY C4	D2	D2	XY D2									
SP 2	A2	A2	XY A2				C1	C1	XY C1	C5	C5	XY C5				E0	E0	XY E0						
SP 3				B0	B0	XY B0	C2	C2	XY C2				D0	D0	XY D0	E1	E1	XY E1						

READS

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21	T22	T23
SP 0	XY 0-3	XY 16-19	XY 32-35	XY 48-51	A0	B1	C3	D1					A0	B1	C3	D1					V 0-3	V 16-19	V 32-35	V 48-51
SP 1	XY 4-7	XY 20-23	XY 36-39	XY 52-55	A1		C4	D2		C0			A1		C4	D2		C0			V 4-7	V 20-23	V 36-39	V 52-55
SP 2	XY 8-11	XY 24-27	XY 40-43	XY 56-59	A2		C5			C1		E0	A2		C5			C1		E0	V 8-11	V 24-27	V 40-43	V 56-59
SP 3	XY 12-15	XY 28-31	XY 44-47	XY 60-63					B0	C2	D0	E1					B0	C2	D0	E1	V 12-15	V 28-31	V 44-47	V 60-63
	XY				P1								P2								VTX			

Figure 6: Interpolation timing diagram

Above is an example of a tile the sequencer might receive from the SC. The write side is how the data get stacked into the XY and IJ buffers, the read side is how the data is passed to the GPRs. The IJ information is packed in the IJ buffer 4 quads at a time or two clocks. The sequencer allows at any given time as many as four quads to interpolate a parameter. They all have to come from the same primitive. Then the sequencer controls the write mask to the GPRs to write the valid data in.

{ISSUE : Do we do the center + centroid approach using both IJ buffers?}

3. Instruction Store

There is going to be only one instruction store for the whole chip. It will contain 4096 instructions of 96 bits each.

It is likely to be a 1 port memory; we use 1 clock to load the ALU instruction, 1 clocks to load the Fetch instruction, 1 clock to load 2 control flow instructions and 1 clock to write instructions.

The instruction store is loaded by the CP thru the register mapped registers.

The next picture shows the various modes the CP can load the memory. The Sequencer has to keep track of the loading modes in order to wrap around the correct boundaries. The wrap-around points are arbitrary and they are specified in the VS_BASE and PIX_BASE control registers. The VS_BASE and PS_BASE context registers are used to specify for each context where its shader is in the instruction memory.

For the Real time commands the story is quite the same but for some small differences. There are no wrap-around points for real time so the driver must be careful not to overwrite regular shader data. The shared code (shared subroutines) uses the same path as real time.

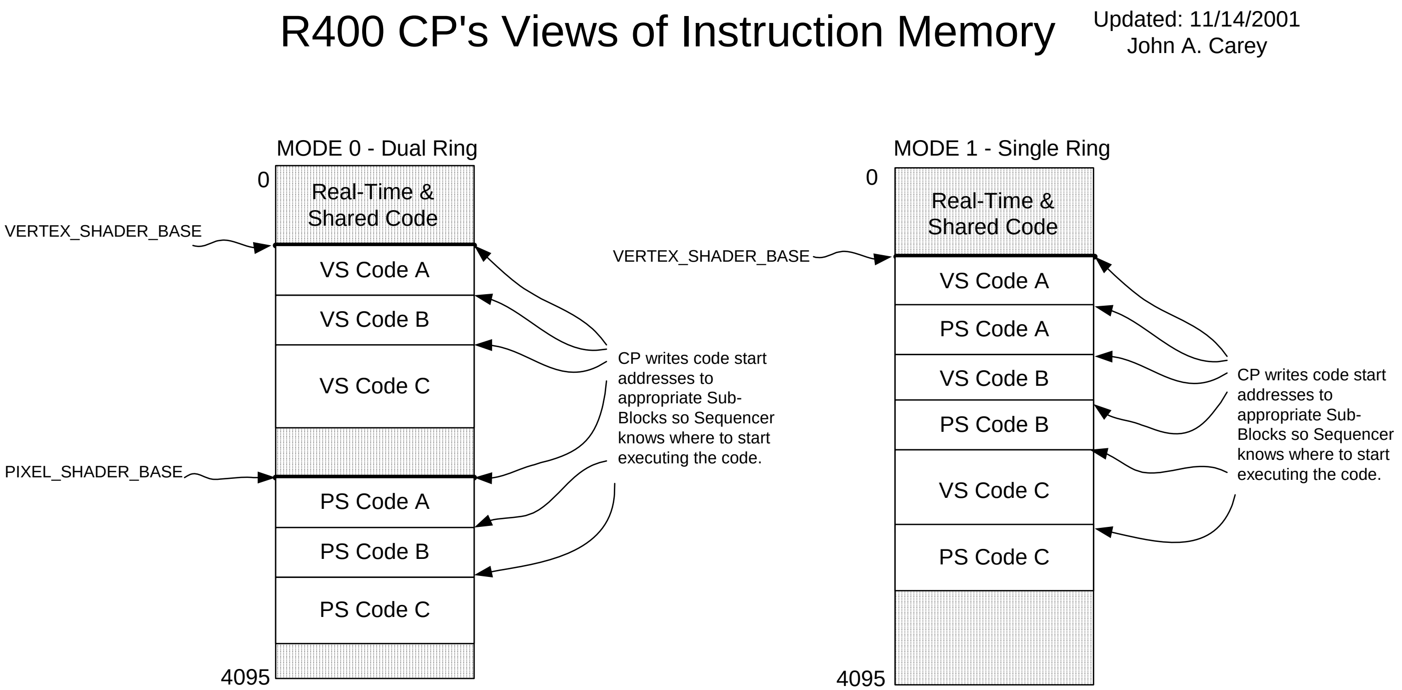


Figure 7: The CP's view of the instruction memory

4. Sequencer Instructions

All control flow instructions and move instructions are handled by the sequencer only. The ALUs will perform NOPs during this time (MOV PV,PV, PS,PS) if they have nothing else to do.

5. Constant Stores

5.1 Memory organizations

A likely size for the ALU constant store is 1024x128 bits. The read BW from the ALU constant store is 128 bits/clock and the write bandwidth is 32 bits/clock (directed by the CP bus size not by memory ports).

The maximum logical size of the constant store for a given shader is 256 constants. Or 512 for the pixel/vertex shader pair. The size of the re-mapping table is 128 lines (each line addresses 4 constants). The write granularity is 4 constants or 512 bits. It takes 16 clocks to write the four constants. Real time requires 256 lines in the physical memory (this is physically register mapped).

The texture state is also kept in a similar memory. The size of this memory is 128x192 bits. The memory thus holds 128 texture states (192 bits per state). The logical size exposes 32 different states total, which are going to be shared between the pixel and the vertex shader. The size of the re-mapping table to for the texture state memory is 32 lines (each line addresses 1 texture state lines in the real memory). The CP write granularity is 1 texture state lines (or 192 bits). The driver sends 512 bits but the CP ignores the top 320 bits. It thus takes 6 clocks to write the texture state. Real time requires 32 lines in the physical memory (this is physically register mapped).

The control flow constant memory doesn't sit behind a renaming table. It is register mapped and thus the driver must reload its content each time there is a ~~state~~ change in the control flow constants. Its size is 320*32 because it must hold 8 copies of the 32 dwords of control flow constants and the loop construct constants must be aligned.

The constant re-mapping tables for texture state and ALU constants are logically register mapped for regular mode and physically register mapped for RT operation.

5.2 Management of the Control Flow Constants

Formatted: Bullets and Numbering

The control flow constants are register mapped, thus the CP writes to the according register to set the constant, the SQ decodes the address and writes to the block pointed by its current base pointer (CF_WR_BASE). On the read side, one level of indirection is used. A register (SQ_CONTEXT_MISC.CF_RD_BASE) keeps the current base pointer to the control flow block. This register is copied whenever there is a state change. Should the CP write to CF after the state change, the base register is updated with the (current pointer number +1)% number of states. This way, if the CP doesn't write to CF the state is going to use the previous CF constants.

~~5.2~~5.3 Management of the re-mapping tables

Formatted: Bullets and Numbering

~~5.2.1~~5.3.1 *R400 Constant management*

The sequencer is responsible to manage two re-mapping tables (one for the constant store and one for the texture state). On a state change (by the driver), the sequencer will broadside copy the contents of its re-mapping tables to a new one. We have 8 different re-mapping tables we can use concurrently.

The constant memory update will be incremental, the driver only need to update the constants that actually changed between the two state changes.

For this model to work in its simplest form, the requirement is that the physical memory MUST be at least twice as large as the logical address space + the space allocated for Real Time. In our case, since the logical address space

is 512 and the reserved RT space can be up to 256 entries, the memory must be of sizes 1280 and above. Similarly the size of the texture store must be of 32*2+32 = 96 entries and above.

~~5.2.2~~5.3.2 Proposal for R400LE constant management

Formatted: Bullets and Numbering

To make this scheme work with only 512+256 = 768 entries, upon reception of a CONTROL packet of state + 1, the sequencer would check for SQ_IDLE and PA_IDLE and if both are idle will erase the content of state to replace it with the new state (this is depicted in Figure 9: De-allocation mechanism~~Figure 9: De-allocation mechanism~~~~Figure 9: De-allocation mechanism~~). Note that in the case a state is cleared a value of 0 is written to the corresponding de-allocation counter location so that when the SQ is going to report a state change, nothing will be de-allocated upon the first report.

The second path sets all context dirty bits that were used in the current state to 1 (thus allowing the new state to reuse these physical addresses if needed).

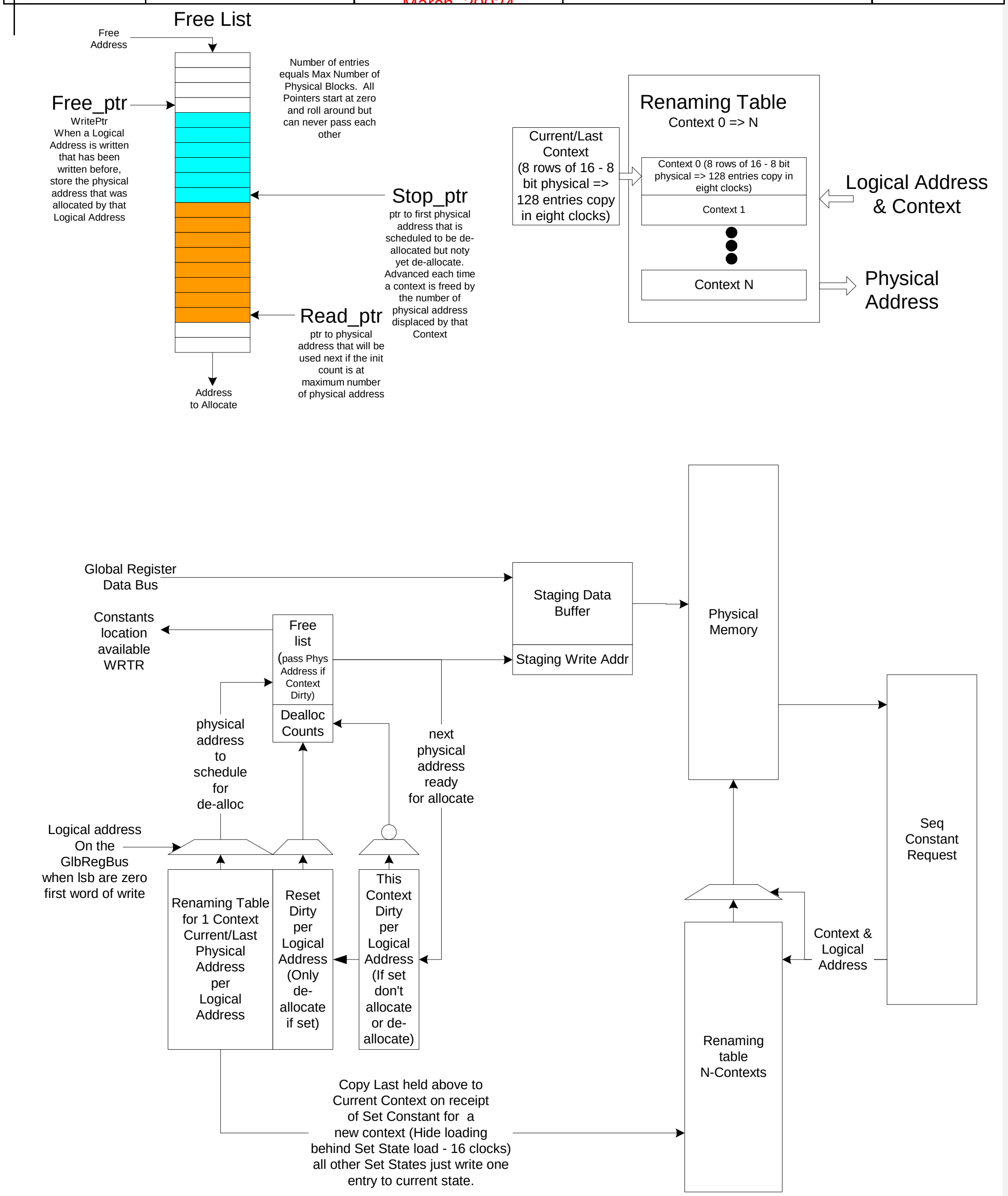

Free List
Free Address
Free_ptr
WritePtr
When a Logical Address is written that has been written before, store the physical address that was allocated by that Logical Address
Number of entries equals Max Number of Physical Blocks. All Pointers start at zero and roll around but can never pass each other
Stop_ptr
ptr to first physical address that is scheduled to be de-allocated but noty yet de-allocate. Advanced each time a context is freed by the number of physical address displaced by that Context
Read_ptr
ptr to physical address that will be used next if the init count is at maximum number of physical address
Address to Allocate
Current/Last Context (8 rows of 16 - 8 bit physical => 128 entries copy in eight clocks)
Renaming Table
Context 0 => N
Context 0 (8 rows of 16 - 8 bit physical => 128 entries copy in eight clocks)
Context 1
Context N
Logical Address & Context
Physical Address
Global Register Data Bus
Staging Data Buffer
Staging Write Addr
Physical Memory
Constants location available WRTR
Free list (pass Phys Address if Context Dirty)
Dealloc Counts
physical address to schedule for de-alloc
next physical address ready for allocate
Logical address On the GlbRegBus when lsb are zero first word of write
Renaming Table for 1 Context Current/Last Physical Address per Logical Address
Reset Dirty per Logical Address (Only de-allocate if set)
This Context Dirty per Logical Address (If set don't allocate or de-allocate)
Seq Constant Request
Context & Logical Address
Renaming table N-Contexts
Copy Last held above to Current Context on receipt of Set Constant for a new context (Hide loading behind Set State load - 16 clocks) all other Set States just write one entry to current state.

Figure 8: Constant management

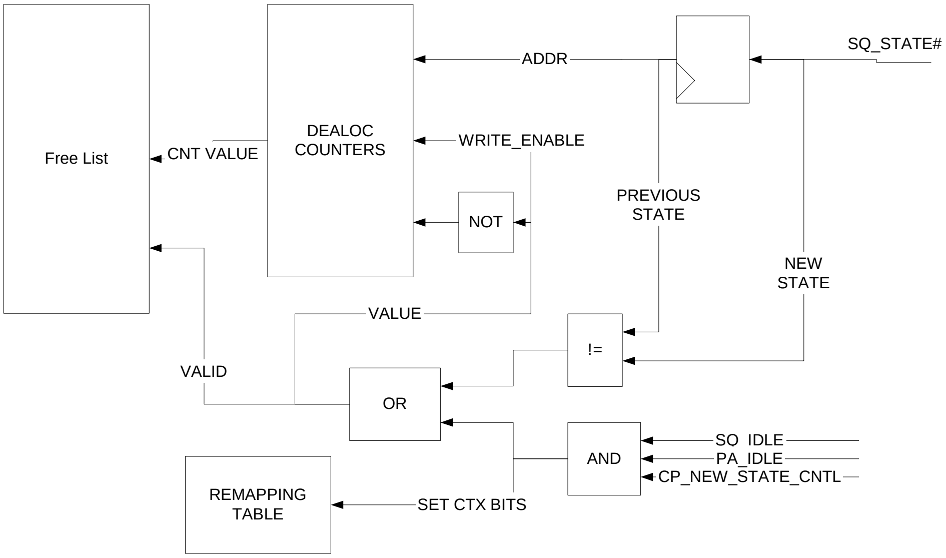


Figure 9: De-allocation mechanism for R400LE

~~5.2.3~~5.3.3 *Dirty bits*

Formatted: Bullets and Numbering

Two sets of dirty bits will be maintained per logical address. The first one will be set to zero on reset and set when the logical address is addressed. The second one will be set to zero whenever a new context is written and set for each address written while in this context. The reset dirty is not set, then writing to that logical address will not require de-allocation of whatever address stored in the renaming table. If it is set and the context dirty is not set, then the physical address store needs to be de-allocated and a new physical address is necessary to store the incoming data. If they are both set, then the data will be written into the physical address held in the renaming for the current logical address. No de-allocation or allocation takes place. This will happen when the driver does a set constant twice to the same logical address between context changes. NOTE: It is important to detect and prevent this, failure to do it will allow multiple writes to allocate all physical memory and thus hang because a context will not fit for rendering to start and thus free up space.

~~5.2.4~~5.3.4 *Free List Block*

Formatted: Bullets and Numbering

A free list block that would consist of a counter (called the IFC or Initial Free Counter) that would reset to zero and incremented every time a chunk of physical memory is used until they have all been used once. This counter would be checked each time a physical block is needed, and if the original ones have not been used up, us a new one, else check the free list for an available physical block address. The count is the physical address for when getting a chunk from the counter.

Storage of a free list big enough to store all physical block addresses.

Maintain three pointers for the free list that are reset to zero. The first one we will call write_ptr. This pointer will identify the next location to write the physical address of a block to be de-allocated. Note: we can never free more physical memory locations than we have. Once recording address the pointer will be incremented to walk the free list like a ring.

The second pointer will be called stop_ptr. The stop_ptr pointer will be advanced by the number of address chunks de-allocates when a context finishes. The address between the stop_ptr and write_ptr cannot be reused because they are still in use. But as soon as the context using then is dismissed the stop_ptr will be advanced.

The third pointer will be called read_ptr. This pointer will point will point to the next address that can be used for allocation as long as the read_ptr does not equal the stop_ptr and the IFC is at its maximum count.

~~5.2.5~~5.3.5 *De-allocate Block*

Formatted: Bullets and Numbering

This block will maintain a free physical address block count for each context. While in current context, a count shall be maintained specifying how many blocks were written into the free list at the write_ptr pointer. This count will be reset upon reset or when this context is active on the back and different than the previous context. It is actually a count of blocks in the previous context that will no longer be used. This count will be used to advance the write_ptr pointer to make available the set of physical blocks freed when the previous context was done. This allows the discard or de-allocation of any number of blocks in one clock.

~~5.2.6~~5.3.6 *Operation of Incremental model*

Formatted: Bullets and Numbering

The basic operation of the model would start with the write_ptr, stop_ptr, read_ptr pointers in the free list set to zero and the free list counter is set to zero. Also all the dirty bits and the previous context will be initialized to zero. When the first set constants happen, the reset dirty bit will not be set, so we will allocate a physical location from the free list counter because its not at the max value. The data will be written into physical address zero. Both the additional copy of the renaming table and the context zeros of the big renaming table will be updated for the logical address that was written by set start with physical address of 0. This process will be repeated for any logical address that are not dirty until the context changes. If a logical address is hit that has its dirty bits set while in the same context, both dirty bits would be set, so the new data will be over-written to the last physical address assigned for this logical address. When the first draw command of the context is detected, the previous context stored in the additional renaming table will be copied to the larger renaming table in the current (new) context location. Then the set constant logical address with be loaded with a new physical address during the copy and if the reset dirty was set, the physical address it replaced in the renaming table would be entered at the write_ptr pointer location on the free list and the write_ptr will be incremented. The de-allocation counter for the previous context (eight) will be incremented. This as set states come in for this context one of the following will happen:

1.) No dirty bits are set for the logical address being updated. A line will be allocated of the free-list counter or the free list at read_ptr pointer if read_ptr != to stop_ptr .
2.) Reset dirty set and Context dirty not set. A new physical address is allocated, the physical address in the renaming table is put on the free list at write_ptr and it is incremented along with the de-allocate counter for the last context.
3.) Context dirty is set then the data will be written into the physical address specified by the logical address.

This process will continue as long as set states arrive. This block will provide backpressure to the CP whenever he has not free list entries available (counter at max and stop_ptr == read_ptr). The command stream will keep a count of contexts of constants in use and prevent more than max constants contexts from being sent.

Whenever a draw packet arrives, the content of the re-mapping table is written to the correct re-mapping table for the context number. Also if the next context uses less constants than the current one all exceeding lines are moved to the free list to be de-allocated later. This happens in parallel with the writing of the re-mapping table to the correct memory.

Now preferable when the constant context leaves the last ALU clause it will be sent to this block and compared with the previous context that left. (Init to zero) If they differ than the older context will no longer be referenced and thus can be de-allocated in the physical memory. This is accomplished by adding the number of blocks freed this context to the stop_ptr pointer. This will make all the physical addresses used by this context available to the read_ptr allocate pointer for future allocation.

This device allows representation of multiple contexts of constants data with N copies of the logical address space. It also allows the second context to be represented as the first set plus some new additional data by just storing the delta's. It allows memory to be efficiently used and when the constants updates are small it can store multiple context. However, if the updates are large, less contexts will be stored and potentially performance will be degraded. Although it will still perform as well as a ring could in this case.

~~5.3~~5.4 Constant Store Indexing

Formatted: Bullets and Numbering

In order to do constant store indexing, the sequencer must be loaded first with the indexes (that come from the GPRs). There are 144 wires from the exit of the SP to the sequencer (9 bits pointers x 16 vertexes/clock). Since the data must pass thru the Shader pipe for the float to fixed conversion, there is a latency of 4 clocks (1 instruction)

between the time the sequencer is loaded and the time one can index into the constant store. The assembly will look like this

```
MOVA  R1.X,R2.X       // Loads the sequencer with the content of R2.X, also copies the content of R2.X into R1.X
NOP                   // latency of the float to fixed conversion
ADD   R3,R4,C0[R2.X]// Uses the state from the sequencer to add R4 to C0[R2.X] into R3
```

Note that we don't really care about what is in the brackets because we use the state from the MOVA instruction. R2.X is just written again for the sake of simplicity and coherency.

The storage needed in the sequencer in order to support this feature is 2*64*9 bits = 1152 bits.

~~5.4~~5.5 Real Time Commands

Formatted: Bullets and Numbering

The real time commands constants are written by the CP using the register mapped registers allocated for RT. It works is the same way than when dealing with regular constant loads BUT in this case the CP is not sending a logical address but rather a physical address and the reads are not passing thru the re-mapping table but are directly read from the memory. The boundary between the two zones is defined by the CONST_EO_RT control register. Similarly, for the fetch state, the boundary between the two zones is defined by the TSTATE_EO_RT control register.

~~5.5~~5.6 Constant Waterfalling

Formatted: Bullets and Numbering

In order to have a reasonable performance in the case of constant store indexing using the address register, we are going to have the possibility of using the physical memory port for read only. This way we can read 1 constant per clock and thus have a worst-case waterfall mode of 1 vertex per clock. There is a small synchronization issue related with this as we need for the SQ to make sure that the constants where actually written to memory (not only sent to the sequencer) before it can allow the first vector of pixels or vertices of the state to go thru the ALUs. To do so, the sequencer keeps 8 bits (one per render state) and sets the bits whenever the last render state is written to memory and clears the bit whenever a state is freed.

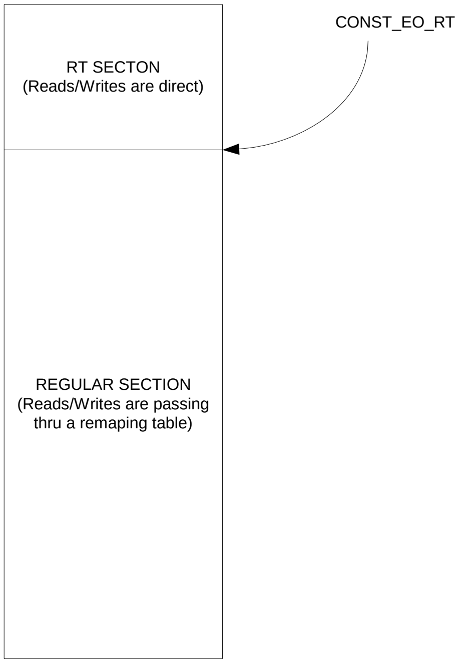


Figure 10: The instruction store

6. Looping and Branches

Loops and branches are planned to be supported and will have to be dealt with at the sequencer level. We plan on supporting constant loops and branches using a control program.

6.1 The controlling state.

The R400 controling state consists of:

Boolean[256:0]
Loop_count[7:0][31:0]
Loop_Start[7:0][31:0]
Loop_Step[7:0][31:0]

That is 256 Booleans and 32 loops.

We have a stack of 4 elements for nested calls of subroutines and 4 loop counters to allow for nested loops.

This state is available on a per shader program basis.

6.2 The Control Flow Program

Examples of control flow programs are located in the R400 programming guide document.

The basic model is as follows:

The render state defined the clause boundaries:

Vertex_shader_fetch[7:0][7:0]	// eight 8 bit pointers to the location where each clauses control program is located
Vertex_shader_alu[7:0][7:0]	// eight 8 bit pointers to the location where each clauses control program is located
Pixel_shader_fetch[7:0][7:0]	// eight 8 bit pointers to the location where each clauses control program is located
Pixel_shader_alu[7:0][7:0]	// eight 8 bit pointers to the location where each clauses control program is located

A pointer value of FF means that the clause doesn't contain any instructions.

The control program for a given clause is executed to completion before moving to another clause, (with the exception of the pick two nature of the alu execution). The control program is the only program aware of the clause boundaries.

The control program has ~~eleven~~ nine basic instructions:

Execute
Conditional_execute
Conditional_Execute_Predicates
Conditional_jump
Conditionnal_Call
Return
Loop_start
Loop_end
~~End_of_clause~~
~~Conditional_End_of_clause~~
NOP

Execute, causes the specified number of instructions in instruction store to be executed.
Conditional_execute checks a condition first, and if true, causes the specified number of instructions in instruction store to be executed.
Loop_start resets the corresponding loop counter to the start value on the first pass after it checks for the end condition and if met jumps over to a specified address.
Loop_end increments (decrements?) the loop counter and jumps back the specified number of instructions.

Conditionnal_Call jumps to an address and pushes the IP counter on the stack if the condition is met. On the return instruction, the IP is popped from the stack.
Conditional_execute_Predicates executes a block of instructions if all bits in the predicate vectors meet the condition.
~~End_of_clause marks the end of a clause.~~
~~Conditional_End_of_clause marks the end of a clause if the condition is met.~~
Conditional_jumps jumps to an address if the condition is met.
NOP is a regular NOP

NOTE THAT ALL JUMPS MUST JUMP TO EVEN CFP ADDRESSES since there are two control flow instructions per memory line. Thus the compiler must insert NOPs where needed to align the jumps on even CFP addresses.

Also if the jump is logically bigger than pshader_cntl_size (or vshader_cntl_size) we break the program (clause) and set the debug registers. If an execute or conditional_execute is lower than cntl_size or bigger than size we also break the program (clause) and set the debug registers.

We have to fit instructions into 48 bits in order to be able to put two control flow instruction per line in the instruction store.

Note that whenever a field is marked as RESERVED, it is assumed that all the bits of the field are cleared (0).

Execute					
47	46… 42	41~~41 … 24~~	40 … 24	23 … 12	11 … 0
Addressing	00001	Last~~RESERVED~~	RESERVED	Instruction count	Exec Address

Execute up to 4k instructions at the specified address in the instruction memory. If Last is set, this is the last group of instructions of the clause.

NOP			
47	46 … 42	41~~41 … 0~~	40 … 0
Addressing	00010	Last~~RESERVED~~	RESERVED

This is a regular NOP. If Last is set, this is the last instruction of the clause.

Conditional_Execute							
47	46 … 42	41	40 … 33	32	31 … 24	23 … 12	11 … 0
Addressing	00011	~~RESERVED~~ Last	Boolean address	Condition	RESERVED	Instruction count	Exec Address

If the specified Boolean (8 bits can address 256 Booleans) meets the specified condition then execute the specified instructions (up to 4k instructions). If Last is set, then if the condition is met, this is the last group of instructions to be executed in the clause. If the condition is not met, we go on to the next control flow instruction.

Conditional_Execute_Predicates								
47	46 … 42	41 ~~… 35~~	40 … 35	34 … 33	32	31 … 24	23 … 12	11 … 0
Addressing	00100	Last ~~RESERVED~~	RESERVED	Predicate vector	Condition	RESERVED	Instruction count	Exec Address

Check the AND/OR of all current predicate bits. If AND/OR matches the condition execute the specified number of instructions. We need to AND/OR this with the kill mask in order not to consider the pixels that aren't valid. If Last is set, then if the condition is met, this is the last group of instructions to be executed in the clause. If the condition is not met, we go on to the next control flow instruction.

Loop_Start				
47	46 … 42	41 … 17	16 … 12	11 … 0
Addressing	00101	RESERVED	loop ID	Jump address

Loop Start. Compares the loop iterator with the end value. If loop condition not met jump to the address. Forward jump only. Also computes the index value. The loop id must match between the start to end, and also indicates which control flow constants should be used with the loop.

Loop_End				
47	46 … 42	41 … 17	16 … 12	11 … 0
Addressing	00110	RESERVED	loop ID	start address

Loop end. Increments the counter by one, compares the loop count with the end value. If loop condition met, continue, else, jump BACK to the start of the loop.

The way this is described does not prevent nested loops, and the inclusion of the loop id make this easy to do.

Conditionnal_Call						
47	46 … 42	41 … 35	34 … 33	32	31 … 12	11 … 0
Addressing	00111	RESERVED	Predicate vector	Condition	RESERVED	Jump address

If the condition is met, jumps to the specified address and pushes the control flow program counter on the stack.

Return		
47	46 … 42	41 … 0
Addressing	01000	RESERVED

Pops the topmost address from the stack and jumps to that address. If nothing is on the stack, the program will just continue to the next instruction.

Conditionnal_Jump							
47	46 … 42	41	40 … 33	32	31	30 … 12	11 … 0
Addressing	01001	RESERVED	Boolean address	Condition	FW only	RESERVED	Jump address

If condition met, jumps to the address. FORWARD jump only allowed if bit 31 set. Bit 31 is only an optimization for the compiler and should NOT be exposed to the API.

~~This is an optimization in the case of very short shaders (where the control flow instruction can't be hidden anymore and thus are not free. In this case, if the condition is met, the clause is ended, else we continue the execution of the clause.~~

~~Marks the end of a clause.~~

To prevent infinite loops, we will keep 9 bits loop iterators instead of 8 (we are only able to loop 256 times). If the counter goes higher than 255 then the loop_end or the loop_start instruction is going to break the loop and set the debug GPRs.

6.3 Data dependant predicate instructions

Data dependant conditionals will be supported in the R400. The only way we plan to support those is by supporting three vector/scalar predicate operations of the form:

PRED_SETE_# - similar to SETE except that the result is 'exported' to the sequencer.
PRED_SETNE_# - similar to SETNE except that the result is 'exported' to the sequencer.
PRED_SETGT_# - similar to SETGT except that the result is 'exported' to the sequencer
PRED_SETGTE_# - similar to SETGTE except that the result is 'exported' to the sequencer

For the scalar operations only we will also support the two following instructions:
PRED_SETE0_# – SETE0
PRED_SETE1_# – SETE1

The export is a single bit - 1 or 0 that is sent using the same data path as the MOVA instruction. The sequencer will maintain 4 sets of 64 bit predicate vectors (in fact 8 sets because we interleave two programs but only 4 will be exposed) and use it to control the write masking. This predicate is not maintained across clause boundaries. The # sign is used to specify which predicate set you want to use 0 thru 3.

Then we have two conditional execute bits. The first bit is a conditional execute “on” bit and the second bit tells us if we execute on 1 or 0. For example, the instruction:

P0_ADD_# R0,R1,R2

Is only going to write the result of the ADD into those GPRs whose predicate bit is 0. Alternatively, P1_ADD_# would only write the results to the GPRs whose predicate bit is set. The use of the P0 or P1 without precharging the sequencer with a PRED instruction is undefined.

{Issue: do we have to have a NOP between PRED and the first instruction that uses a predicate?}

6.4 HW Detection of PV,PS

Because of the control program, the compiler cannot detect statically dependant instructions. In the case of non-masked writes and subsequent reads the sequencer will insert uses of PV,PS as needed. This will be done by comparing the read address and the write address of consecutive instructions. For masked writes, the sequencer will insert NOPs wherever there is a dependant read/write.

The sequencer will also have to insert NOPs between PRED_SET and MOVA instructions and their uses.

6.5 Register file indexing

Because we can have loops in fetch clause, we need to be able to index into the register file in order to retrieve the data created in a fetch clause loop and use it into an ALU clause. The instruction will include the base address for register indexing and the instruction will contain these controls:

Bit7	Bit 6	
0	0	'absolute register'
0	1	'relative register'
1	0	'previous vector'
1	1	'previous scalar'

In the case of an absolute register we just take the address as is. In the case of a relative register read we take the base address and we add to it the loop_index and this becomes our new address that we give to the shader pipe.

The sequencer is going to keep a loop index computed as such:

Index = Loop_iterator*Loop_step + Loop_start.

We loop until loop_iterator = loop_count. Loop_step is a signed value [-128…127]. The computed index value is a 10 bit counter that is also signed. Its real range is [-256,256]. The tenth bit is only there so that we can provide an out of range value to the “indexing logic” so that it knows when the provided index is out of range and thus can make the necessary arrangements.

6.6 Predicated Instruction support for Texture clauses

For texture clauses, we support the following optimization: we keep 1 bit (thus 4 bits for the four predicate vectors) per predicate vector in the reservation stations. A value of 1 means that one ore more elements in the vector have a value of one (thus we have to do the texture fetches for the whole vector). A value of 0 means that no elements in the vector have his predicate bit set and we can thus skip over the texture fetch. **We have to make sure the invalid pixels aren't considered with this optimization.**

6.7 Debugging the Shaders

In order to be able to debug the pixel/vertex shaders efficiently, we provide 2 methods.

6.7.1 *Method 1: Debugging registers*

Current plans are to expose 2 debugging, or error notification, registers:
1. address register where the first error occurred
2. count of the number of errors

The sequencer will detect the following groups of errors:
- count overflow
- constant indexing overflow
- register indexing overflow

Compiler recognizable errors:
- jump errors
    relative jump address > size of the control flow program
- call stack
    call with stack full
    return with stack empty

A jump error will always cause the program to break. In this case, a break means that a clause will halt execution, but allowing further clauses to be executed.

With all the other errors, program can continue to run, potentially to worst-case limits. The program will only break if the DB_PROB_BREAK register is set.

If indexing outside of the constant or the register range, causing an overflow error, the hardware is specified to return the value with an index of 0. This could be exploited to generate error tokens, by reserving and initializing the 0th register (or constant) for errors.

{ISSUE : Interrupt to the driver or not?}

6.7.2 *Method 2: Exporting the values in the GPRs (12)*

The sequencer will have a count register and an address register for this mode and 3 bits per clause specifying the execution mode for each clause. The modes can be :

1) Normal
2) Debug Kill
3) Debug Addr + Count

Under the normal mode execution follows the normal course. Under the kill mode, all control flow instructions are executed but all normal shader instructions of the clause are replaced by NOPs. Only debug_export instructions of clause 7 will be executed under the debug kill setting. Under the other mode, normal execution is done until we reach an address specified by the address register and instruction count (useful for loops) specified by the count register. After we have hit the instruction n times (n=count) we switch the clause to the kill mode.

Under the debug mode (debug kill OR debug Addr + count), it is assumed that clause 7 is always exporting 12 debug vectors and that all other exports to the SX block (position, color, z, ect) will been turned off (changed into NOPs) by the sequencer (even if they occur before the address stated by the ADDR debug register).

7. Pixel Kill Mask

A vector of 64 bits is kept by the sequencer per group of pixels/vertices. Its purpose is to optimize the texture fetch requests and allow the shader pipe to kill pixels using the following instructions:

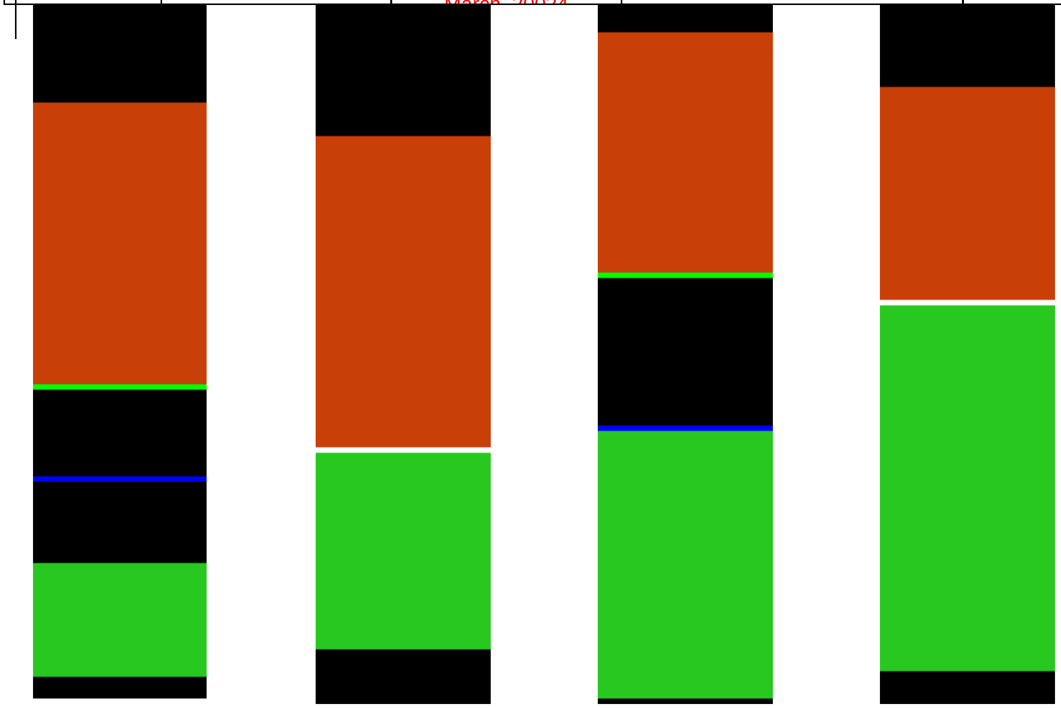
MASK_SETE
MASK_SETNE
MASK_SETGT
MASK_SETGTE

8. Multipass vertex shaders (HOS)

Multipass vertex shaders are able to export from the 6 last clauses but to memory ONLY.

9. Register file allocation

The register file allocation for vertices and pixels can either be static or dynamic. In both cases, the register file in managed using two round robins (one for pixels and one for vertices). In the dynamic case the boundary between pixels and vertices is allowed to move, in the static case it is fixed to 128-VERTEX_REG_SIZE for vertices and PIXEL_REG_SIZE for pixels.



Above is an example of how the algorithm works. Vertices come in from top to bottom; pixels come in from bottom to top. Vertices are in orange and pixels in green. The blue line is the tail of the vertices and the green line is the tail of the pixels. Thus anything between the two lines is shared. When pixels meets vertices the line turns white and the boundary is static until both vertices and pixels share the same "unallocated bubble". Then the boundary is allowed to move again. The numbering of the GPRs starts from the bottom of the picture at index 0 and goes up to the top at index 127.

10. Fetch Arbitration

The fetch arbitration logic chooses one of the 8 potentially pending fetch clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute. Once chosen, the clause state machine will send one 2x2 fetch per clock (or 4 fetches in one clock every 4 clocks) until all the fetch instructions of the clause are sent. This means that there cannot be any dependencies between two fetches of the same clause.

The arbitrator will not wait for the fetches to return prior to selecting another clause for execution. The fetch pipe will be able to handle up to X(?) in flight fetches and thus there can be a fair number of active clauses waiting for their fetch return data.

11. ALU Arbitration

ALU arbitration proceeds in almost the same way than fetch arbitration. The ALU arbitration logic chooses one of the 8 potentially pending ALU clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute. There are two ALU arbiters, one for the even clocks and one for the odd clocks. For example, here is the sequencing of two interleaved ALU clauses (E and O stands for Even and Odd sets of 4 clocks):

Einst0 Oinst0 Einst1 Oinst1 Einst2 Oinst2 Einst0 Oinst3 Einst1 Oinst4 Einst2 Oinst0…
Proceeding this way hides the latency of 8 clocks of the ALUs. Also note that the interleaving also occurs across clause boundaries.

12. Handling Stalls

When the output file is full, the sequencer prevents the ALU arbitration logic from selecting the last clause (this way nothing can exit the shader pipe until there is place in the output file. If the packet is a vertex packet and the position buffer is full (POS_FULL) then the sequencer also prevents a thread from entering the exporting clause (3?). The sequencer will set the OUT_FILE_FULL signal n clocks before the output file is actually full and thus the ALU arbiter will be able read this signal and act accordingly by not preventing exporting clauses to proceed.

13. Content of the reservation station FIFOs

The reservation FIFOs contain the state of the vector of pixels and vertices. We have two sets of those: one for pixels, and one for vertices. They contain 3 bits of Render State 7 bits for the base address of the GPRs, some bits for LOD correction and coverage mask information in order to fetch fetch for only valid pixels, the quad address.

14. The Output File

The output file is where pixels are put before they go to the RBs. The write BW to this store is 256 bits/clock. Just before this output file are staging registers with write BW 512 bits/clock and read BW 256 bits/clock. The staging registers are 4x128 (and there are 16 of those on the whole chip).

15. IJ Format

The IJ information sent by the PA is of this format on a per quad basis:

We have a vector of IJ's (one IJ per pixel at the centroid of the fragment or at the center of the pixel depending on the mode bit). The interpolation is done at a different precision across the 2x2. The upper left pixel's parameters are always interpolated at full 20x24 mantissa precision. Then the result of the interpolation along with the difference in IJ in reduced precision is used to interpolate the parameter for the other three pixels of the 2x2. Here is how we do it:

Assuming P0 is the interpolated parameter at Pixel 0 having the barycentric coordinates I(0), J(0) and so on for P1,P2 and P3. Also assuming that A is the parameter value at V0 (interpolated with I), B is the parameter value at V1 (interpolated with J) and C is the parameter value at V2 (interpolated with (1-I-J).

$$\Delta 01I = I(1) - I(0)$$
$$\Delta 01J = J(1) - J(0)$$
$$\Delta 02I = I(2) - I(0)$$
$$\Delta 02J = J(2) - J(0)$$
$$\Delta 03I = I(3) - I(0)$$
$$\Delta 03J = J(3) - J(0)$$

P0	P1
P2	P3

$$P0 = C + I(0)*(A-C) + J(0)*(B-C)$$
$$P1 = P0 + \Delta 01I*(A-C) + \Delta 01J*(B-C)$$
$$P2 = P0 + \Delta 02I*(A-C) + \Delta 02J*(B-C)$$
$$P3 = P0 + \Delta 03I*(A-C) + \Delta 03J*(B-C)$$

P0 is computed at 20x24 mantissa precision and P1 to P3 are computed at 8X24 mantissa precision. So far no visual degradation of the image was seen using this scheme.

Multiplies (Full Precision): 2
Multiplies (Reduced precision): 6
Subtracts 19x24 (Parameters): 2

Adds: 8

FORMAT OF P0's IJ : Mantissa 20 Exp 4 for I + Sign
Mantissa 20 Exp 4 for J + Sign

FORMAT of Deltas (x3):Mantissa 8 Exp 4 for I + Sign
Mantissa 8 Exp 4 for J + Sign

Total number of bits : 20*2 + 8*6 + 4*8 + 4*2 = 128

All numbers are kept using the un-normalized floating point convention: if exponent is different than 0 the number is normalized if not, then the number is un-normalized. The maximum range for the IJs (Full precision) is +/- 63 and the range for the Deltas is +/- 127.

15.1 Interpolation of constant attributes

Because of the floating point imprecision, we need to take special provisions if all the interpolated terms are the same or if two of the barycentric coordinates are the same.

We start with the premise that if A = B and B = C and C = A, then P0,1,2,3 = A. Since one or more of the IJ terms may be zero, so we extend this to:

```
if (A=B and B=C and C=A)
   P0,1,2,3 = A;
else if ((I = 0) or (J = 0)) and
          ((J = 0) or (1-I-J = 0)) and
          ((1-J-I = 0) or (I = 0))) {
                if(I != 0) {
                    P0 = A;
                } else if(J != 0) {
                    P0 = B;
                } else {
                    P0 = C;
               }
             //rest of the quad interpolated normally
}
else
{
             normal interpolation
}
```

16. Staging Registers

In order for the reuse of the vertices to be 14, the sequencer will have to re-order the data sent IN ORDER by the VGT for it to be aligned with the parameter cache memory arrangement. Given the following group of vertices sent by the VGT:

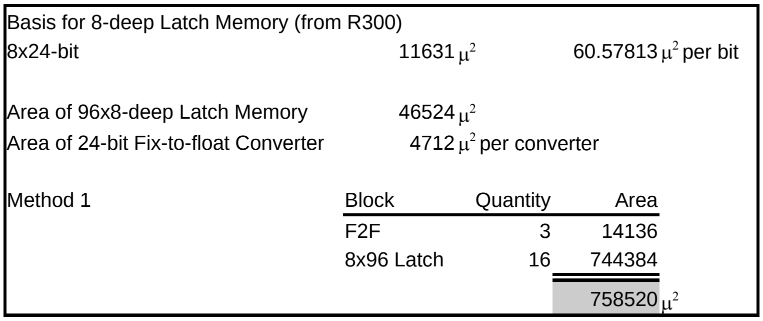
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 || 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 || 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 || 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63

The sequencer will re-arrange them in this fashion:

0 1 2 3 16 17 18 19 32 33 34 35 48 49 50 51 || 4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55 || 8 9 10 11 24 25 26 27 40 41 42 43 56 57 58 59 || 12 13 14 15 28 29 30 31 44 45 46 47 60 61 62 63

The || markers show the SP divisions. In the event a shader pipe is broken, the VGT will send padding to account for the missing pipe. For example, if SP1 is broken, vertices 4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55 will still be sent by the VGT to the SQ **BUT** will not be processed by the SP and thus should be considered invalid (by the SU and VGT).

The most straightforward, *non-compressed* interface method would be to convert, in the VGT, the data to 32-bit floating point prior to transmission to the VSISRs. In this scenario, the data would be transmitted to (and stored in) the VSISRs in full 32-bit floating point. This method requires three 24-bit fixed-to-float converters in the VGT. Unfortunately, it also requires and additional 3,072 bits of storage across the VSISRs. This interface is illustrated in Figure 12~~Figure 12Figure 12~~. The area of the fixed-to-float converters and the VSISRs for this method is roughly estimated as 0.759sqmm using the R300 process. The gate count estimate is shown in Figure 11~~Figure 11Figure 11~~.



Basis for 8-deep Latch Memory (from R300)			
8x24-bit	11631 μ^2		60.57813 μ^2 per bit
Area of 96x8-deep Latch Memory	46524 μ^2		
Area of 24-bit Fix-to-float Converter	4712 μ^2 per converter		
Method 1	Block	Quantity	Area
	F2F	3	14136
	8x96 Latch	16	744384
			758520 μ^2

Figure 11:Area Estimate for VGT to Shader Interface

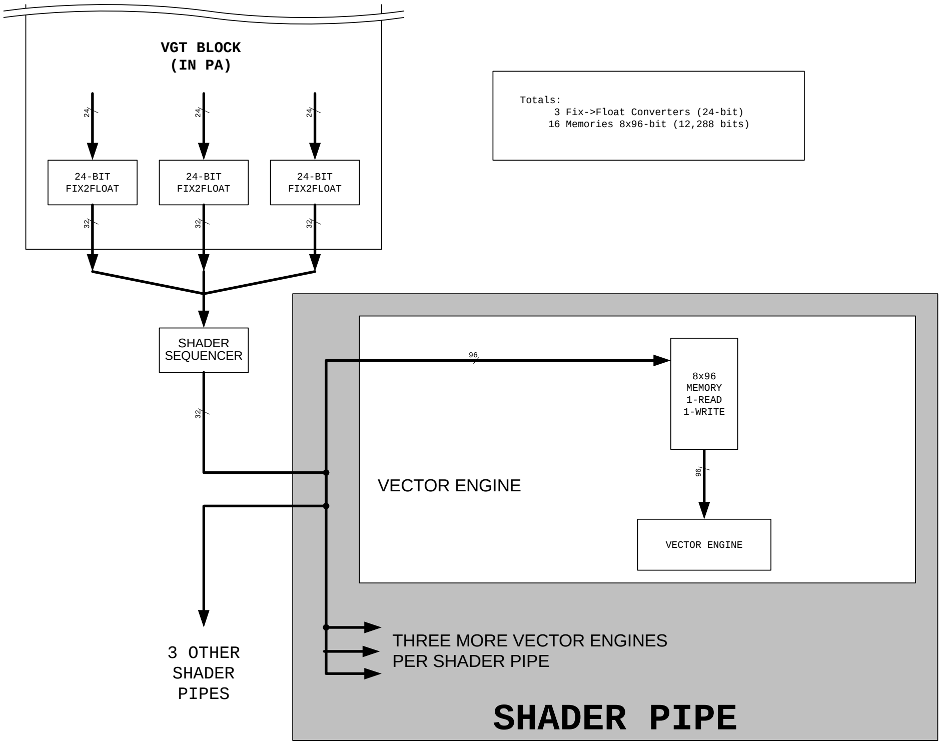


Figure 12:VGT to Shader Interface

17. The parameter cache

The parameter cache is where the vertex shaders export their data. It consists of 16 128x128 memories (1R/1W). The reuse engine will make it so that all vertexes of a given primitive will hit different memories. The allocation method for these memories is a simple round robin. The parameter cache pointers are mapped in the following way: 4MSBs are the memory number and the 7 LSBs are the address within this memory.

MEMORY NUMBER 4 bits	ADDRESS 7 bits

The PA generates the parameter cache addresses as the positions come from the SQ. All it needs to do is keep a Current_Location pointer (7 bits only) and as the positions comes increment the memory number. When the memory number field wraps around, the PA increments the Current_Location by VS_EXPORT_COUNT_7 (a snooped register from the SQ). As an example, say the memories are all empty to begin with and the vertex shader is exporting 8 parameters per vertex (VS_EXPORT_COUNT_7 = 8). The first position received is going to have the PC address 00000000000 the second one 00010000000, third one 00100000000 and so on up to 11110000000. Then the next position received (the 17th) is going to have the address 00000001000, the 18th 00010001000, the 19th 00100001000 and so on. The Current_location is NEVER reset BUT on chip resets. The only thing to be careful about is that if the SX doesn't send you a full group of positions (<64) then you need to fill the address space so that the next group starts correctly aligned (for example if you receive only 33 positions then you need to add 2*VS_EXPORT_COUNT_7to Current_Location and reset the memory count to 0 before the next vector begins).

18. Vertex position exporting

On clause 3 the vertex shader can export to the PA both the vertex position and the point sprite. It can also do so at clause 7 if not done at clause 3. The storage needed to perform the position export is at least 64x128 memories for the position and 64x32 memories for the sprite size. It is going to be taken in the pixel output fifo from the SX blocks. The clause where the position export occurs is specified by the EXPORT_LATE register. If turned on, it means that the export is going to occur at ALU clause 7 if unset position export occurs at clause 3.

19. Exporting Arbitration

Here are the rules for co-issuing exporting ALU clauses.

1) Position exports and position exports cannot be co-issued.

All other types of exports can be co-issued as long as there is place in the receiving buffer.

{ISSUE: Do we move the parameter caches to the SX?}

20. Export Types

The export type (or the location where the data should be put) is specified using the destination address field in the ALU instruction. Here is a list of all possible export modes:

20.1 Vertex Shading

0:15	- 16 parameter cache
16:31	- Empty (Reserved?)
32:43	- 12 vertex exports to the frame buffer and index
44:47	- Empty
48:59	- 12 debug export (interpret as normal vertex export)
60	- export addressing mode
61	- Empty
62	- position
63	- sprite size export that goes with position export (point_h,point_w,edgeflag,misc)

20.2 Pixel Shading

0	- Color for buffer 0 (primary)
1	- Color for buffer 1
2	- Color for buffer 2
3	- Color for buffer 3
4:7	- Empty
8	- Buffer 0 Color/Fog (primary)
9	- Buffer 1 Color/Fog
10	- Buffer 2 Color/Fog
11	- Buffer 3 Color/Fog
12:15	- Empty
16:31	- Empty (Reserved?)
32:43	- 12 exports for multipass pixel shaders.
44:47	- Empty
48:59	- 12 debug exports (interpret as normal pixel export)
60	- export addressing mode
61:62	- Empty
63	- Z for primary buffer (Z exported to 'alpha' component)

21. Special Interpolation modes

21.1 Real time commands

We are unable to use the parameter memory since there is no way for a command stream to write into it. Instead we need to add three 16x128 memories (one for each of three vertices x 16 interpolants). These will be mapped onto the register bus and written by type 0 packets, and output to the the parameter busses (the sequencer and/or PA need to be able to address the reatime parameter memory as well as the regular parameter store. For higher performance we should be able able to view them as two banks of 16 and do double buffering allowing one to be loaded, while the other is rasterized with. Most overlay shaders will need 2 or 4 scalar coordinates, one option might be to restrict the memory to 16x64 or 32x64 allowing only two interpolated scalars per cycle, the only problem I see with this is, if we view support for 16 vector-4 interpolants important (true only if we map Microsoft's high priority stream to the realtime stream), then the PA/sequencer need to support a realtime-specific mode where we need to address 32 vectors of parameters instead of 16. This mode is triggered by the primitive type: REAL TIME. The actual memories are in the in the SX blocks. The parameter data memories are hooked on the RBBM bus and are loaded by the CP using register mapped memory.

21.2 Sprites/ XY screen coordinates/ FB information

When working with sprites, one may want to overwrite the parameter 0 with SC generated data. Also, XY screen coordinates may be needed in the shader program. This functionality is controlled by the gen_I0 register (in SQ) in conjunction with the SND_XY register (in SC). Also it is possible to send the faceness information (for OGL front/back special operations) to the shader using the same control register. Here is a list of all the modes and how they interact together:

Gen_st is a bit taken from the interface between the SC and the SQ. This is the MSB of the primitive type. If the bit is set, it means we are dealing with Point AA, Line AA or sprite and in this case the vertex values are going to generated between 0 and 1.

Param_Gen_I0 disable, snd_xy disable, no gen_st – I0 = No modification
Param_Gen_I0 disable, snd_xy disable, gen_st – I0 = No modification
Param_Gen_I0 disable, snd_xy enable, no gen_st – I0 = No modification
Param_Gen_I0 disable, snd_xy enable, gen_st – I0 = No modification
Param_Gen_I0 enable, snd_xy disable, no gen_st – I0 = garbage, garbage, garbage, faceness
Param_Gen_I0 enable, snd_xy disable, gen_st – I0 = garbage, garbage, s, t
Param_Gen_I0 enable, snd_xy enable, no gen_st – I0 = screen x, screen y, garbage, faceness
Param_Gen_I0 enable, snd_xy enable, gen_st – I0 = screen x, screen y, s, t

21.3 Auto generated counters

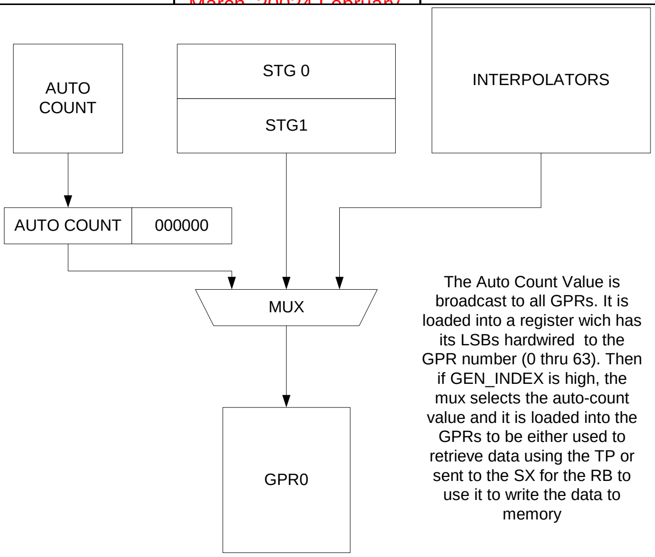
In the cases we are dealing with multipass shaders, the sequencer is going to generate a vector count to be able to both use this count to write the 1st pass data to memory and then use the count to retrieve the data on the 2nd pass. The count is always generated in the same way but it is passed to the shader in a slightly different way depending on the shader type (pixel or vertex). This is toggled on and off using the GEN_INDEX register. The sequencer is going to keep two counters, one for pixels and one for vertices. Every time a full vector of vertices or pixels is written to the GPRs the counter is incremented. Every time a state change is detected, the corresponding counter is reset. While there is only one count broadcast to the GPRs, the LSB are hardwired to specific values making the index different for all elements in the vector.

21.3.1 *Vertex shaders*

In the case of vertex shaders, if GEN_INDEX is set, the data will be put into the x field of the third register (it means that the compiler must allocate 3 GPRs in all multipass vertex shader modes).

21.3.2 *Pixel shaders*

In the case of pixel shaders, if GEN_INDEX is set and Param_Gen_I0 is enabled, the data will be put in the x field of the 2nd register (R1.x), else if GEN_INDEX is set the data will be put into the x field of the 1st register (R0.x).



The Auto Count Value is broadcast to all GPRs. It is loaded into a register wich has its LSBs hardwired to the GPR number (0 thru 63). Then if GEN_INDEX is high, the mux selects the auto-count value and it is loaded into the GPRs to be either used to retrieve data using the TP or sent to the SX for the RB to use it to write the data to memory

Figure 13: GPR input mux Control

22. State management

Every clock, the sequencer will report to the CP the oldest states still in the pipe. These are the states of the programs as they enter the last ALU clause.

22.1 Parameter cache synchronization

In order for the sequencer not to begin a group of pixels before the associated group of vertices has finished, the sequencer will keep a 6 bit count per state (for a total of 8 counters). These counters are initialized to 0 and every time a vertex shader exports its data TO THE PARAMETER CACHE, the corresponding pointer is incremented. When the SC sends a new vector of pixels with the SC_SQ_new_vector bit asserted, the sequencer will first check if the count is greater than 0 before accepting the transmission (it will in fact accept the transmission but then lower its ready to receive). Then the sequencer waits for the count to go to one and decrements it. The sequencer can then issue the group of pixels to the interpolators. Every time the state changes, the new state counter is initialized to 0.

23. XY Address imports

The SC will be able to send the XY addresses to the GPRs. It does so by interleaving the writes of the IJs (to the IJ buffer) with XY writes (to the XY buffer). Then when writing the data to the GPRs, the sequencer is going to interpolate the IJ data or pass the XY data thru a Fix→float converter and expander and write the converted values to the GPRs. The Xys are currently SCREEN SPACE COORDINATES. The values in the XY buffers will wrap. See section 21.2 for details on how to control the interpolation in this mode.

23.1 Vertex indexes imports

In order to import vertex indexes, we have 16 8x96 staging registers. These are loaded one line at a time by the VGT block (96 bits). They are loaded in floating point format and can be transferred in 4 or 8 clocks to the GPRs.

24. Registers

24.1 Control

REG_DYNAMIC	Dynamic allocation (pixel/vertex) of the register file on or off.
REG_SIZE_PIX	Size of the register file's pixel portion (minimal size when dynamic allocation turned on)
REG_SIZE_VTX	Size of the register file's vertex portion (minimal size when dynamic allocation turned on)
ARBITRATION_POLICY	policy of the arbitration between vertexes and pixels
INST_STORE_ALLOC	interleaved, separate
INST_BASE_VTX	start point for the vertex instruction store (RT always ends at vertex_base and Begins at 0)
INST_BASE_PIX	start point for the pixel shader instruction store
ONE_THREAD	debug state register. Only allows one program at a time into the GPRs
ONE_ALU	debug state register. Only allows one ALU program at a time to be executed (instead of 2)
INSTRUCTION	This is where the CP puts the base address of the instruction writes and type (auto-incremented on reads/writes) Register mapped
CONSTANTS	512*4 ALU constants + 32*6 Texture state 32 bits registers (logically mapped)
CONSTANTS_RT	256*4 ALU constants + 32*6 texture states? (physically mapped)
CONSTANT_EO_RT	This is the size of the space reserved for real time in the constant store (from 0 to CONSTANT_EO_RT). The re-mapping table operates on the rest of the memory
TSTATE_EO_RT	This is the size of the space reserved for real time in the fetch state store (from 0 to TSTATE_EO_RT). The re-mapping table operates on the rest of the memory
EXPORT_LATE	Controls whether or not we are exporting position from clause 3. If set, position exports occur at clause 7.

24.2 Context

VS_FETCH_{0…7}	eight 8 bit pointers to the location where each clauses control program is located
VS_ALU_{0…7}	eight 8 bit pointers to the location where each clauses control program is located
PS_FETCH_{0…7}	eight 8 bit pointers to the location where each clauses control program is located
PS_ALU_{0…7}	eight 8 bit pointers to the location where each clauses control program is located
PS_BASE	base pointer for the pixel shader in the instruction store
VS_BASE	base pointer for the vertex shader in the instruction store
VS_CF_SIZE	size of the vertex shader (# of instructions in control program/2)
PS_CF_SIZE	size of the pixel shader (# of instructions in control program/2)
PS_SIZE	size of the pixel shader (cntl+instructions)
VS_SIZE	size of the vertex shader (cntl+instructions)
PS_NUM_REG	number of GPRs to allocate for pixel shader programs
VS_NUM_REG	number of GPRs to allocate for vertex shader programs
PARAM_SHADE	One 16 bit register specifying which parameters are to be gouraud shaded (0 = flat, 1 = gouraud)
PROVO_VERT	0 : vertex 0, 1: vertex 1, 2: vertex 2, 3: Last vertex of the primitive
PARAM_WRAP	64 bits: for which parameters (and channels (xyzw)) do we do the cyl wrapping (0=linear, 1=cylindrical).
PS_EXPORT_MODE	0xxxx : Normal mode 1xxxx : Multipass mode If normal, bbbz where bbb is how many colors (0-4) and z is export z or not If multipass 1-12 exports for color.
VS_EXPORT_MASK	which of the last 6 ALU clauses is exporting (multipass only)
VS_EXPORT_MODE	0: position (1 vector), 1: position (2 vectors), 3:multipass
VS_EXPORT _COUNT_{0…6}	Six 4 bit counters representing the # of interpolated parameters exported in clause 7 (located in VS_EXPORT_COUNT_6) OR # of exported vectors to memory per clause in multipass mode (per clause)
PARAM_GEN_I0	Do we overwrite or not the parameter 0 with XY data and generated T and S values

GEN_INDEX	Auto generates an address from 0 to XX. Puts the results into R0-1 for pixel shaders and R2 for vertex shaders
CONST_BASE_VTX (9 bits)	Logical Base address for the constants of the Vertex shader
CONST_BASE_PIX (9 bits)	Logical Base address for the constants of the Pixel shader
CONST_SIZE_PIX (8 bits)	Size of the logical constant store for pixel shaders
CONST_SIZE_VTX (8 bits)	Size of the logical constant store for vertex shaders
INST_PRED_OPTIMIZE	Turns on the predicate bit optimization (if of, conditional_execute_predicates is always executed).
CF_BOOLEANS	256 boolean bits
CF_LOOP_COUNT	32x8 bit counters (number of times we traverse the loop)
CF_LOOP_START	32x8 bit counters (init value used in index computation)
CF_LOOP_STEP	32x8 bit counters (step value used in index computation)

25. DEBUG Registers

25.1 Context

DB_PROB_ADDR	instruction address where the first problem occurred
DB_PROB_COUNT	number of problems encountered during the execution of the program
DB_PROB_BREAK	break the clause if an error is found.
DB_INST_COUNT	instruction counter for debug method 2
DB_BREAK_ADDR	break address for method number 2
DB_CLAUSE _MODE_ALU_{0…7}	clause mode for debug method 2 (0: normal, 1: addr, 2: kill)
DB_CLAUSE _MODE_FETCH_{0…7}	clause mode for debug method 2 (0: normal, 1: addr, 2: kill)

25.2 Control

DB_ALUCST_MEMSIZE	Size of the physical ALU constant memory
DB_TSTATE_MEMSIZE	Size of the physical texture state memory

26. Interfaces

26.1 External Interfaces

Whenever an x is used, it means that the bus is broadcast to all units of the same name. For example, if a bus is named SQ→SPx it means that SQ is going to broadcast the same information to all SP instances.

26.1.1 *SC to SQ : IJ Control bus*

This is the control information sent to the sequencer in order to control the IJ fifos and all other information needed to execute a shader program on the sent pixels. This information is sent over 2 clocks, if SENDXY is asserted the next control packet is going to be ignored and XY information is going to be sent on the IJ bus (for the quads that where just sent). All pixels from the group of quads are from the same primitive, all quads of a vector are from the same render state.

Name	Direction	Bits	Description
SC_SQ_q_wr_mask	SC→SQ	4	Quad Write mask left to right
SC_SQ_lod_correct	SC→SQ	24	LOD correction per quad (6 bits per quad)
SC_SQ_param_ptr0	SC→SQ	11	P Store pointer for vertex 0
SC_SQ_param_ptr1	SC→SQ	11	P Store pointer for vertex 1
SC_SQ_param_ptr2	SC→SQ	11	P Store pointer for vertex 2
SC_SQ_end_of_vect	SC→SQ	1	End of the vector
SC_SQ_store_dealloc	SC→SQ	1	Deallocation token for the P Store
SC_SQ_state	SC→SQ	3	State/constant pointer
SC_SQ_valid_pixel	SC→SQ	16	Valid bits for all pixels
SC_SQ_null_prim	SC→SQ	1	Null Primitive (for PC deallocation purposes)
SC_SQ_end_of_prim	SC→SQ	1	End Of the primitive
SC_SQ_send_xy	SC→SQ	1	Sending XY information [XY information is going to be sent on the next clock]
SC_SQ_prim_type	SC→SQ	3	Real time command need to load tex cords from alternate buffer. Line AA, Point AA and Sprite reads their parameters from GEN_T and GEN_S GPRs. 000 : Normal 011 : Real Time 100 : Line AA 101 : Point AA 110 : Sprite
SC_SQ_new_vector	SC→SQ	1	This primitive comes from a new vector of vertices. Make sure that the corresponding vertex shader has finished before starting the group of pixels.
SC_SQ_RTRn	SQ→SC	1	Stalls the PA in n clocks
SC_SQ_RTS	SC→SQ	1	SC ready to send data

26.1.2 *SQ to SP: Interpolator bus*

Name	Direction	Bits	Description
SQ_SPx_interp_prim_type	SQ→SPx	3	Type of the primitive 000 : Normal 011 : Real Time 100 : Line AA 101 : Point AA 110 : Sprite
SQ_SPx_interp_ijline	SQ→SPx	2	Line in the IJ/XY buffer to use to interpolate
SQ_SPx_interp_buff_swap	SQ→SPx	1	Swap the IJ/XY buffers at the end of the interpolation
SQ_SPx_interp_gen_I0	SQ→SPx	1	Generate I0 or not. This tells the interpolators not to use the parameter cache but rather overwrite the data with interpolated 1 and 0. Overwrite if gen_I0 is high.

26.1.3 *SQ to SX: Interpolator bus*

Name	Direction	Bits	Description
SQ_SPx_interp_flat_vtx	SQ→SPx	2	Provoking vertex for flat shading
SQ_SPx_interp_flat_gouraud	SQ→SPx	1	Flat or gouraud shading
SQ_SPx_interp_cyl_wrap	SQ→SPx	4	Wich channel needs to be cylindrical wrapped
SQ_SXx_mux0	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_mux1	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_mux2	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_RT_switch	SQ→SXx	1	Selects between RT and Normal data

~~26.1.4SQ to SP: Parameter Cache Read control bus~~

Formatted: Bullets and Numbering

~~The four following interfaces (SQ→SP, SQ→SX,SP→SX and SX→Interpolators) are all SYNCHRONIZED together.~~

~~26.1.4 SQ to SX: Parameter Cache Mux control Bus~~

~~26.1.6~~26.1.4 SQ to SP: Staging Register Data

Formatted: Bullets and Numbering

This is a broadcast bus that sends the VSISR information to the staging registers of the shader pipes.

Name	Direction	Bits	Description
SQ_SPx_vgt_vsisr_data	SQ→SPx	96	Pointers of indexes or HOS surface information
SQ_SPx_vgt_vsisr_double	SQ→SPx	1	0: Normal 96 bits per vert 1: double 192 bits per vert
SQ_SP0_data_valid	SQ→SP0	1	Data is valid
SQ_SP1_data_valid	SQ→SP1	1	Data is valid
SQ_SP2_data_valid	SQ→SP2	1	Data is valid
SQ_SP3_data_valid	SQ→SP3	1	Data is valid

~~26.1.7~~26.1.5 PA to SQ : Vertex interface

Formatted: Bullets and Numbering

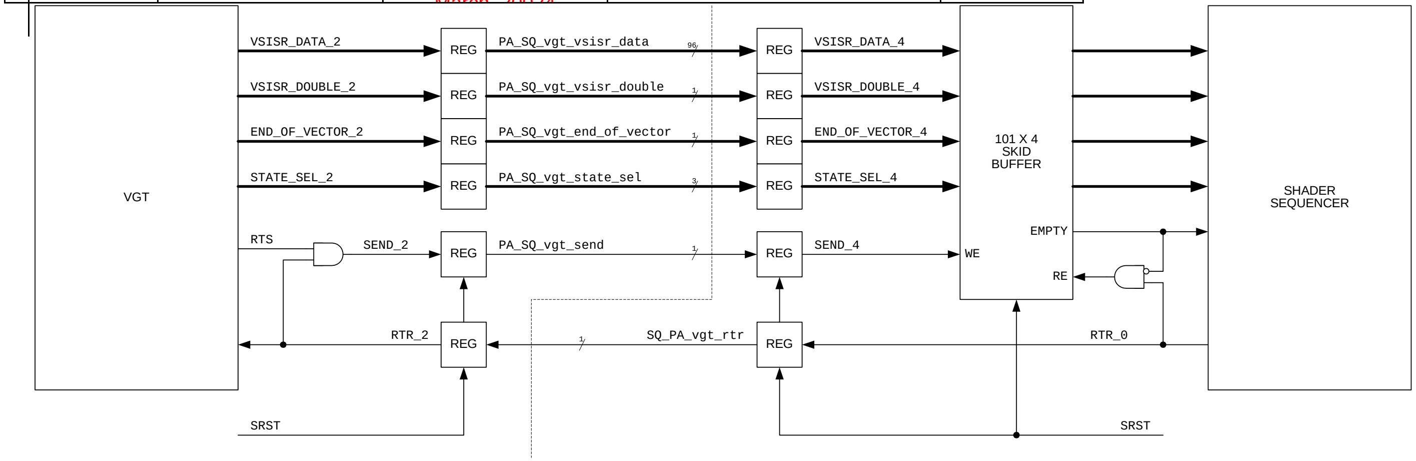
~~26.1.7.1~~26.1.5.1 Interface Signal Table

The area difference between the two methods is not sufficient to warrant complicating the interface or the state requirements of the VSISRs. **Therefore, the POR for this interface is that the VGT will transmit the data to the VSISRs (via the Shader Sequencer) in full, 32-bit floating-point format.** The VGT can transmit up to six 32-bit floating-point values to each VSISR where four or more values require two transmission clocks. The data bus is 96 bits wide.

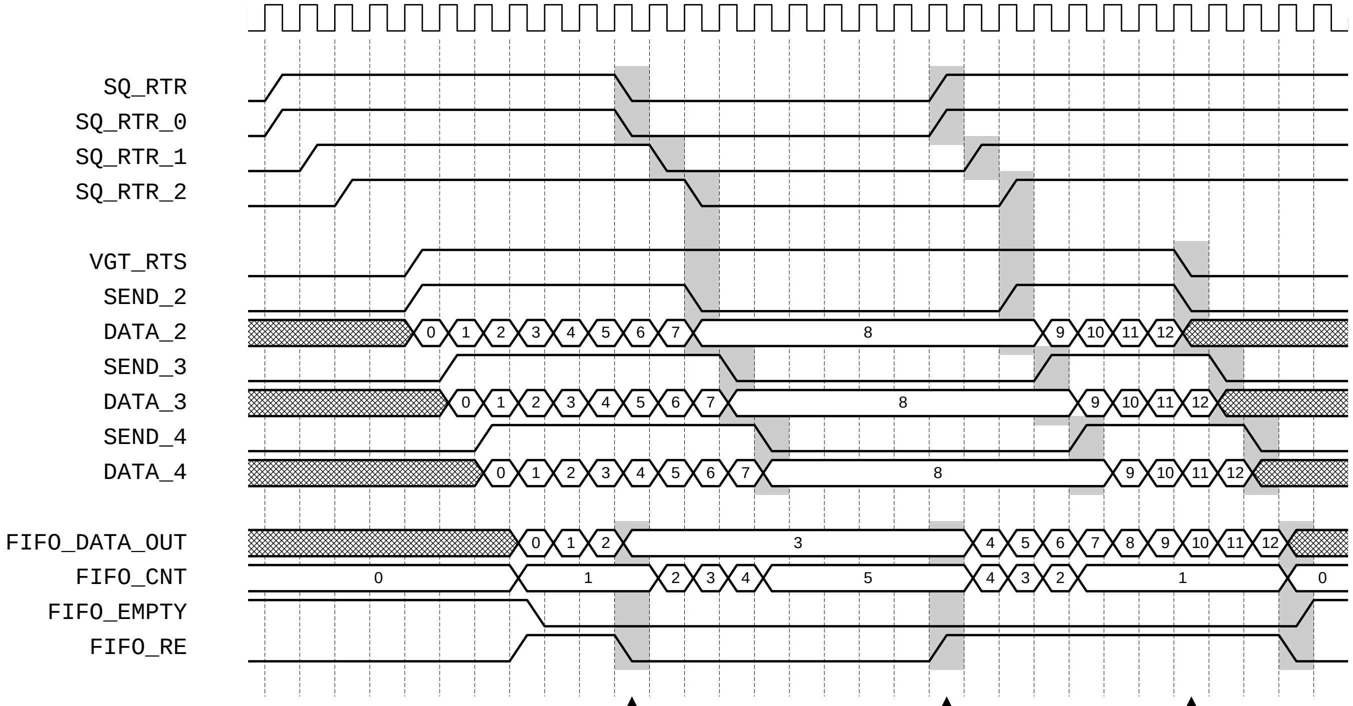
Name	Bits	Description
PA_SQ_vgt_vsisr_data	96	Pointers of indexes or HOS surface information
PA_SQ_vgt_vsisr_double	1	0: Normal 96 bits per vert 1: double 192 bits per vert
PA_SQ_vgt_end_of_vector	1	Indicates the last VSISR data set for the current process vector (for double vector data, "end_of_vector" is set on the second vector)
PA_SQ_vgt_vsisr_valid	1	Vsisr data is valid
PA_SQ_vgt_state	3	Render State (6*3+3 for constants). This signal is guaranteed to be correct when “PA_SQ_vgt_end_of_vector” is high.
PA_SQ_vgt_send	1	Data on the VGT_SQ is valid receive (see write-up for standard R400 SEND/RTR interface handshaking)
SQ_PA_vgt_rtr	1	Ready to receive (see write-up for standard R400 SEND/RTR interface handshaking)

~~26.1.7.2~~26.1.5.2 Interface Diagrams

Formatted: Bullets and Numbering



VGT
VSISR_DATA_2
VSISR_DOUBLE_2
END_OF_VECTOR_2
STATE_SEL_2
RTS
SEND_2
RTR_2
SRST
REG
PA_SQ_vgt_vsisr_data
96
PA_SQ_vgt_vsisr_double
1
PA_SQ_vgt_end_of_vector
1
PA_SQ_vgt_state_sel
3
PA_SQ_vgt_send
1
SQ_PA_vgt_rtr
1
VSISR_DATA_4
VSISR_DOUBLE_4
END_OF_VECTOR_4
STATE_SEL_4
SEND_4
101 X 4
SKID
BUFFER
EMPTY
WE
RE
RTR_0
SRST
SHADER
SEQUENCER



RECEIVER STOPS TRANSMISSION

RECEIVER RE-STARTS TRANSMISSION

SENDER STOPS TRANSMISSION

Figure 1. Detailed Logical Diagram for PA_SQ_vgt Interface.

~~26.1.8~~26.1.6 SQ to CP: State report

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_CP_vrtx_ state	SEQ→CP	3	Oldest vertex state still in the pipe
SQ_CP_pix_state	SEQ→CP	3	Oldest pixel state still in the pipe

~~26.1.9~~26.1.7 SQ to SX: Control bus

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SXx_exp_Pixel	SQ→SXx	1	1: Pixel 0: Vertex
SQ_SXx_exp_start	SQ→SXx	1	Raised to indicate that the SQ is starting an exporting clause
SQ_SXx_exp_Clause	SQ→SXx	3	Clause number, which is needed for vertex clauses
SQ_SXx_exp_State	SQ→SXx	3	State ID~~, which is needed for vertex clauses~~
SQ_SXx_exp_VDest	SQ→SXx	6	Export Destination
SQ_SXx_exp_exportID	SQ→SXx	1	ALU ID

These fields are sent synchronously with SP export data, described in SP0→SX0 interface
~~{ISSUE: Where are the PC pointers}~~

~~26.1.10~~26.1.8 SX to SQ : Output file control

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SXx_SQ_Export_count_rdy	SXx→SQ	1	Raised by SX0 to indicate that the following two fields reflect the result of the most recent export
SXx_SQ_Export_Position	SXx→SQ	1	Specifies whether there is room for another position.
SXx_SQ_Export_Buffer	SXx→SQ	7	Specifies the space available in the output buffers. 0: buffers are full 1: 2K-bits available (32-bits for each of the 64 pixels in a clause) ... 64: 128K-bits available (16 128-bit entries for each of 64 pixels) 65-127: RESERVED

~~26.1.11~~26.1.9 SQ to TP: Control bus

Formatted: Bullets and Numbering

Once every clock, the fetch unit sends to the sequencer on which clause it is now working and if the data in the GPRs is ready or not. This way the sequencer can update the fetch counters for the reservation station fifos. The sequencer also provides the instruction and constants for the fetch to execute and the address in the register file where to write the fetch return data.

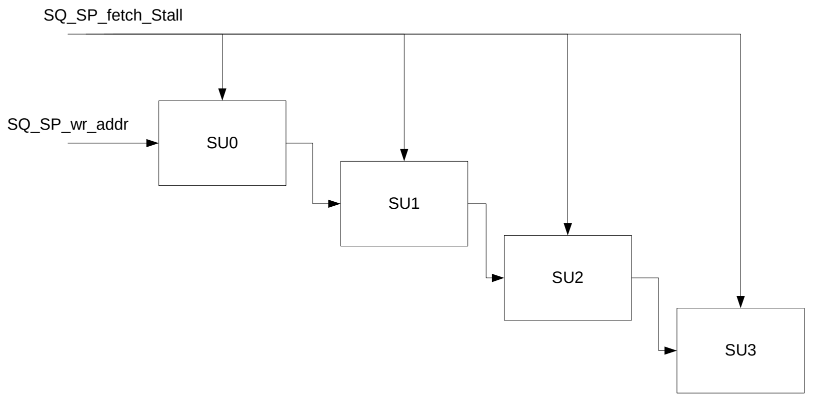
Name	Direction	Bits	Description
TPx_SQ_data_rdy	TPx→ SQ	1	Data ready
TPx_SQ_clause_num	TPx→ SQ	3	Clause number
TPx_SQ_Type	TPx→ SQ	1	Type of data sent (0:PIXEL, 1:VERTEX)
SQ_TPx_const	SQ→TPx	48	Fetch state sent over 4 clocks (192 bits total)
SQ_TPx_instuct	SQ→TPx	24	Fetch instruction sent over 4 clocks
SQ_TPx_end_of_clause	SQ→TPx	1	Last instruction of the clause
SQ_TPx_Type	SQ→TPx	1	Type of data sent (0:PIXEL, 1:VERTEX)
SQ_TPx_phase	SQ→TPx	2	Write phase signal
SQ_TP0_lod_correct	SQ→TP0	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP0_pmask	SQ→TP0	4	Pixel mask 1 bit per pixel
SQ_TP1_lod_correct	SQ→TP1	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP1_pmask	SQ→TP1	4	Pixel mask 1 bit per pixel
SQ_TP2_lod_correct	SQ→TP2	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP2_pmask	SQ→TP2	4	Pixel mask 1 bit per pixel

SQ_TP3_lod_correct	SQ→TP3	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP3_pmask	SQ→TP3	4	Pixel mask 1 bit per pixel
SQ_TPx_clause_num	SQ→TPx	3	Clause number
SQ_TPx_write_gpr_index	SQ->TPx	7	Index into Register file for write of returned Fetch Data

Formatted: Bullets and Numbering

~~26.1.12~~26.1.10 TP to SQ: Texture stall

The TP sends this signal to the SQ when its input buffer is full. The SQ is going to send it to the SP X clocks after reception (maximum of 3 clocks of pipeline delay).



Name	Direction	Bits	Description
TP_SQ_fetch_stall	TP→ SQ	1	Do not send more texture request if asserted

Formatted: Bullets and Numbering

~~26.1.13~~26.1.11 SQ to SP: Texture stall

Name	Direction	Bits	Description
SQ_SPx_fetch_stall	SQ→SPx	1	Do not send more texture request if asserted

Formatted: Bullets and Numbering

~~26.1.14~~26.1.12 SQ to SP: GPR, Parameter cache control and auto counter

Name	Direction	Bits	Description
SQ_SPx_wr_addr	SQ→SPx	7	Write address
SQ_SPx_gpr_rd_addr	SQ→SPx	7	Read address
SQ_SPx_gpr_re_addr	SQ→SPx	1	Read Enable
SQ_SPx_gpr_we_addr	SQ→SPx	1	Write Enable for the GPRs
SQ_SPx_gpr_phase_mux	SQ→SPx	2	The phase mux (arbitrates between inputs, ALU SRC reads and writes)
SQ_SPx_channel_mask	SQ→SPx	4	The channel mask
SQ_SP0_pixel_mask	SQ→SP0	4	The pixel mask
SQ_SP1_pixel_mask	SQ→SP1	4	The pixel mask
SQ_SP2_pixel_mask	SQ→SP2	4	The pixel mask
SQ_SP3_pixel_mask	SQ→SP3	4	The pixel mask
SQ_SPx_pc_we_addr	SQ→SPx	1	Write Enable for the parameter caches
SQ_SPx_gpr_input_mux	SQ→SPx	2	When the phase mux selects the inputs this tells from which source to read from: Interpolated data, VTX0, VTX1, autogen counter.
SQ_SPx_index_count	SQ→SPx	12?	Index count, common for all shader pipes

~~26.1.15~~26.1.13 SQ to SPx: Instructions

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SPx_instruct_start	SQ→SPx	1	Instruction start
SQ_SP_instruct	SQ→SPx	21	Transferred over 4 cycles 0: SRC A Select 2:0 SRC A Argument Modifier 3:3 SRC A swizzle 11:4 ~~Unused~~ VectorDst ~~20~~17:12 Unused 20:18 -- - 1: SRC B Select 2:0 SRC B Argument Modifier 3:3 SRC B swizzle 11:4 ScalarDst 17:12 Unused 20:18 ~~Unused 20:12~~ -- - 2: SRC C Select 2:0 SRC C Argument Modifier 3:3 SRC C swizzle 11:4 Unused 20:12 -- - 3: Vector Opcode 4:0 Scalar Opcode 10:5 Vector Clamp 11:11 Scalar Clamp 12:12 Vector Write Mask 16:13 Scalar Write Mask 20:17
SQ_SPx_stall	SQ→SPx	1	Stall signal
SQ_SPx_export_count	SQ→SPx	3	Each set of four pixels or vectors is exported over eight clocks. This field specifies where the SP is in that sequence.
SQ_SPx_export_last	SQ→SPx	1	Asserted on the first shader count of the last export of the clause
SQ_SP0_export_pvalid	SQ→SP0	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP0_export_wvalid	SQ→SP0	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors
SQ_SP1_ export_pvalid	SQ→SP1	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP1_ export_wvalid	SQ→SP1	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors
SQ_SP2_ export_pvalid	SQ→SP2	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP2_ export_wvalid	SQ→SP2	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors

SQ_SP3_ export_pvalid	SQ→SP3	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP3_ export_wvalid	SQ→SP3	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors

~~26.1.16~~26.1.14 *SP to SQ: Constant address load/ Predicate Set*

Formatted: Bullets and Numbering

Name	**Direction**	**Bits**	**Description**
SP0_SQ_const_addr	SP0→SQ	36	Constant address load / predicate vector load (4 bits only) to the sequencer
SP0_SQ_valid	SP0→SQ	1	Data valid
SP1_SQ_const_addr	SP1→SQ	36	Constant address load / predicate vector load (4 bits only) to the sequencer
SP1_SQ_valid	SP1→SQ	1	Data valid
SP2_SQ_const_addr	SP2→SQ	36	Constant address load / predicate vector load (4 bits only) to the sequencer
SP2_SQ_valid	SP2→SQ	1	Data valid
SP3_SQ_const_addr	SP3→SQ	36	Constant address load / predicate vector load (4 bits only) to the sequencer
SP3_SQ_valid	SP3→SQ	1	Data valid

~~26.1.17~~26.1.15 *SQ to SPx: constant broadcast*

Formatted: Bullets and Numbering

Name	**Direction**	**Bits**	**Description**
SQ_SPx_constant	SQ→SPx	128	Constant broadcast

~~26.1.18~~26.1.16 *SP0 to SQ: Kill vector load*

Formatted: Bullets and Numbering

Name	**Direction**	**Bits**	**Description**
SP0_SQ_kill_vect	SP0→SQ	4	Kill vector load
SP1_SQ_kill_vect	SP1→SQ	4	Kill vector load
SP2_SQ_kill_vect	SP2→SQ	4	Kill vector load
SP3_SQ_kill_vect	SP3→SQ	4	Kill vector load

~~26.1.19~~26.1.17 *SQ to CP: RBBM bus*

Formatted: Bullets and Numbering

Name	**Direction**	**Bits**	**Description**
SQ_RBB_rs	SQ→CP	1	Read Strobe
SQ_RBB_rd	SQ→CP	32	Read Data
SQ_RBBM_nrtrtr	SQ→CP	1	Optional
SQ_RBBM_rtr	SQ→CP	1	Real-Time (Optional)

~~26.1.20~~26.1.18 *CP to SQ: RBBM bus*

Formatted: Bullets and Numbering

Name	**Direction**	**Bits**	**Description**
rbbm_we	CP→SQ	1	Write Enable
rbbm_a	CP→SQ	15	Address -- Upper Extent is TBD (16:2)
rbbm_wd	CP→SQ	32	Data
rbbm_be	CP→SQ	4	Byte Enables
rbbm_re	CP→SQ	1	Read Enable
rbb_rs0	CP→SQ	1	Read Return Strobe 0
rbb_rs1	CP→SQ	1	Read Return Strobe 1
rbb_rd0	CP→SQ	32	Read Data 0
rbb_rd1	CP→SQ	32	Read Data 0
RBBM_SQ_soft_reset	CP→SQ	1	Soft Reset

27. Examples of program executions

27.1.1 *Sequencer Control of a Vector of Vertices*

1. PA sends a vector of 64 vertices (actually vertex indices – 32 bits/index for 2048 bit total) to the RE's Vertex FIFO
   - state pointer as well as tag into position cache is sent along with vertices
   - space was allocated in the position cache for transformed position before the vector was sent
   - **also before the vector is sent to the RE, the CP has loaded the global instruction store with the vertex shader program (using the MH?)**
   - The vertex program is assumed to be loaded when we receive the vertex vector.
     - the SEQ then accesses the IS base for this shader using the local state pointer (provided to all sequencers by the RBBM when the CP is done loading the program)
2. SEQ arbitrates between the Pixel FIFO and the Vertex FIFO – basically the Vertex FIFO always has priority
   - at this point the vector is removed from the Vertex FIFO
   - the arbiter is not going to select a vector to be transformed if the parameter cache is full unless the pipe as nothing else to do (ie no pixels are in the pixel fifo).
3. SEQ allocates space in the SP register file for index data plus GPRs used by the program
   - the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer that came down with the vertices
   - SEQ will not send vertex data until space in the register file has been allocated
4. SEQ sends the vector to the SP register file over the RE_SP interface (which has a bandwidth of 2048 bits/cycle)
   - the 64 vertex indices are sent to the 64 register files over 4 cycles
     - RF0 of SU0, SU1, SU2, and SU3 is written the first cycle
     - RF1 of SU0, SU1, SU2, and SU3 is written the second cycle
     - RF2 of SU0, SU1, SU2, and SU3 is written the third cycle
     - RF3 of SU0, SU1, SU2, and SU3 is written the fourth cycle
   - the index is written to the least significant 32 bits **(floating point format?) (what about compound indices)** of the 128-bit location within the register file (w); the remaining data bits are set to zero (x, y, z)
5. SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of fetch state machine 0, or TSM0 FIFO)
   - the control packet contains the state pointer, the tag to the position cache and a register file base pointer.
6. TSM0 accepts the control packet and fetches the instructions for fetch clause 0 from the global instruction store
   - TSM0 was first selected by the TSM arbiter before it could start
7. all instructions of fetch clause 0 are issued by TSM0
8. the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)
   - TSM0 does not wait for requests made to the Fetch Unit to complete; it passes the register file write index for the fetch data to the TU, which will write the data to the RF as it is received
   - once the TU has written all the data to the register files, it increments a counter that is associated with ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go ahead start to execute the ALU clause
9. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store
10. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of fetch state machine 1, or TSM1 FIFO)
11. the control packet continues to travel down the path of reservation stations until all clauses have been executed
    - position can be exported in ALU clause 3 (or 4?); the data (and the tag) is sent over a position bus (which is shared with all four shader pipes) back to the PA's position cache
    - A parameter cache pointer is also sent along with the position data. This tells to the PA where the data is going to be in the parameter cache.
      - there is a position export FIFO in the SP that buffers position data before it gets sent back to the PA

    - the ASM arbiter will prevent a packet from starting an exporting clause if the position export FIFO is full
- parameter data is exported in clause 7 (as well as position data if it was not exported earlier)
    - parameter data is sent to the Parameter Cache over a dedicated bus
    - the SEQ allocates storage in the Parameter Cache, and the SEQ deallocates that space when there is no longer a need for the parameters (it is told by the PA when using a token).
    - the ASM arbiter will prevent a packet from starting on ASM7 if the parameter cache (or the position buffer if position is being exported) is full

12. after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program

27.1.2 *Sequencer Control of a Vector of Pixels*

1. **As with vertex shader programs, pixel shaders are loaded into the global instruction store by the CP**
    - At this point it is assumed that the pixel program is loaded into the instruction store and thus ready to be read.
2. the RE's Pixel FIFO is loaded with the barycentric coordinates for pixel quads by the detailed walker
    - the state pointer and the LOD correction bits are also placed in the Pixel FIF0
    - the Pixel FIFO is wide enough to source four quad's worth of barycentrics per cycle
3. SEQ arbitrates between Pixel FIFO and Vertex FIFO – when there are no vertices pending OR there is no space left in the register files for vertices, the Pixel FIFO is selected
4. SEQ allocates space in the SP register file for all the GPRs used by the program
    - the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer
    - SEQ will not allow interpolated data to be sent to the shader until space in the register file has been allocated
5. SEQ controls the transfer of interpolated data to the SP register file over the RE_SP interface (which has a bandwidth of 2048 bits/cycle). See interpolated data bus diagrams for details.
6. SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of fetch state machine 0, or TSM0 FIFO)
    - note that there is a separate set of reservation stations/arbiters/state machines for vertices and for pixels
    - the control packet contains the state pointer, the register file base pointer, and the LOD correction bits
    - all other information (such as quad address for example) travels in a separate FIFO
7. TSM0 accepts the control packet and fetches the instructions for fetch clause 0 from the global instruction store
    - TSM0 was first selected by the TSM arbiter before it could start
8. all instructions of fetch clause 0 are issued by TSM0
9. the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)
    - TSM0 does not wait for fetch requests made to the Fetch Unit to complete; it passes the register file write index for the fetch data to the TU, which will write the data to the RF as it is received
    - once the TU has written all the data for a particular clause to the register files, it increments a counter that is associated with the ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go ahead and pop the FIFO and start to execute the ALU clause
10. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store
11. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of fetch state machine 1, or TSM1 FIFO)
12. the control packet continues to travel down the path of reservation stations until all clauses have been executed
    - pixel data is exported in the last ALU clause (clause 7)
        - it is sent to an output FIFO where it will be picked up by the render backend
        - the ASM arbiter will prevent a packet from starting on ASM7 if the output FIFO is full
13. after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program

27.1.3 *Notes*

14. The state machines and arbiters will operate ahead of time so that they will be able to immediately start the real threads or stall.
15. The register file base pointer for a vector needs to travel with the vector through the reservation stations, but the instruction store base pointer does not – this is because the RF pointer is different for all threads, but the IS pointer is only different for each state and thus can be accessed via the state pointer.

28. Open issues

Need to do some testing on the size of the register file as well as on the register file allocation method (dynamic VS static).

Saving power?

Parameter caches in SX?

Using both IJ buffers for center + centroid interpolation?

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 2015~~18~~ March, 20024 March	GEN-CXXXXX-REVA	1 of 50

Author: Laurent Lefebvre

Issue To:

Copy No:

R400 Sequencer Specification

SQ

Version 1.9~~8~~

Overview: This is an architectural specification for the R400 Sequencer block (SEQ). It provides an overview of the required capabilities and expected uses of the block. It also describes the block interfaces, internal sub-blocks, and provides internal state diagrams.

AUTOMATICALLY UPDATED FIELDS:
Document Location: C:\perforce\r400\doc_lib\design\blocks\sq\R400_Sequencer.doc
Current Intranet Search Title: R400 Sequencer Specification

APPROVALS	
Name/Dept	Signature/Date

Remarks:

THIS DOCUMENT CONTAINS CONFIDENTIAL INFORMATION THAT COULD BE SUBSTANTIALLY DETRIMENTAL TO THE INTEREST OF ATI TECHNOLOGIES INC. THROUGH UNAUTHORIZED USE OR DISCLOSURE.

Table Of Contents

1. OVERVIEW .... **8~~6~~**
1.1 Top Level Block Diagram .... 10~~8~~
1.2 Data Flow graph (SP) .... 12~~10~~
1.3 Control Graph .... 13~~11~~
2. INTERPOLATED DATA BUS .... **13~~11~~**
3. INSTRUCTION STORE .... **16~~14~~**
4. SEQUENCER INSTRUCTIONS .... **18~~16~~**
5. CONSTANT STORES .... **18~~16~~**
5.1 Memory organizations .... 18~~16~~
5.2 Management of the Control Flow Constants .... 18~~16~~
5.3 Management of the re-mapping tables .... 18~~16~~
5.3.1 R400 Constant management .... 18~~16~~
5.3.2 Proposal for R400LE constant management .... 19~~17~~
5.3.3 Dirty bits .... 21~~19~~
5.3.4 Free List Block .... 21~~19~~
5.3.5 De-allocate Block .... 22~~20~~
5.3.6 Operation of Incremental model .... 22~~20~~
5.4 Constant Store Indexing .... 22~~20~~
5.5 Real Time Commands .... 23~~21~~
5.6 Constant Waterfalling .... 23~~21~~
6. LOOPING AND BRANCHES .... **24~~22~~**
6.1 The controlling state. .... 24~~22~~
6.2 The Control Flow Program .... 24~~22~~
6.3 Data dependant predicate instructions .... 26~~24~~
6.4 HW Detection of PV,PS .... 27~~25~~
6.5 Register file indexing .... 27~~25~~
6.6 Predicated Instruction support for Texture clauses .... 27~~25~~
6.7 Debugging the Shaders .... 28~~25~~
6.7.1 Method 1: Debugging registers .... 28~~25~~
6.7.2 Method 2: Exporting the values in the GPRs (12) .... 28~~26~~
7. PIXEL KILL MASK .... **28~~26~~**
8. MULTIPASS VERTEX SHADERS (HOS) .... **29~~26~~**
9. REGISTER FILE ALLOCATION .... **29~~26~~**
10. FETCH ARBITRATION .... **30~~28~~**
11. ALU ARBITRATION .... **30~~28~~**
12. HANDLING STALLS .... **31~~29~~**
13. CONTENT OF THE RESERVATION STATION FIFOS .... **31~~29~~**
14. THE OUTPUT FILE .... **31~~29~~**
15. IJ FORMAT .... **31~~29~~**
15.1 Interpolation of constant attributes .... 32~~30~~
16. STAGING REGISTERS .... **32~~30~~**
17. THE PARAMETER CACHE .... **34~~32~~**

18. VERTEX POSITION EXPORTING ...... 35~~32~~
19. EXPORTING ARBITRATION ...... 35~~32~~
20. EXPORT TYPES ...... 35~~32~~
20.1 Vertex Shading ...... 35~~32~~
20.2 Pixel Shading ...... 35~~33~~
21. SPECIAL INTERPOLATION MODES ...... 36~~33~~
21.1 Real time commands ...... 36~~33~~
21.2 Sprites/ XY screen coordinates/ FB information ...... 36~~33~~
21.3 Auto generated counters ...... 36~~34~~
21.3.1 Vertex shaders ...... 36~~34~~
21.3.2 Pixel shaders ...... 36~~34~~
22. STATE MANAGEMENT ...... 37~~34~~
22.1 Parameter cache synchronization ...... 37~~34~~
23. XY ADDRESS IMPORTS ...... 37~~35~~
23.1 Vertex indexes imports ...... 37~~35~~
24. REGISTERS ...... 38~~35~~
24.1 Control ...... 38~~35~~
24.2 Context ...... 38~~35~~
25. DEBUG REGISTERS ...... 39~~36~~
25.1 Context ...... 39~~36~~
25.2 Control ...... 39~~36~~
26. INTERFACES ...... 39~~36~~
26.1 External Interfaces ...... 39~~36~~
26.1.1 SC to SQ : IJ Control bus ...... 39~~37~~
26.1.2 SQ to SP: Interpolator bus ...... 40~~37~~
26.1.3 SQ to SX: Interpolator bus ...... 40~~37~~
26.1.4 SQ to SP: Staging Register Data ...... 41~~38~~
26.1.5 PA to SQ : Vertex interface ...... 41~~38~~
26.1.6 SQ to CP: State report ...... 44~~41~~
26.1.7 SQ to SX: Control bus ...... 44~~41~~
26.1.8 SX to SQ : Output file control ...... 44~~41~~
26.1.9 SQ to TP: Control bus ...... 44~~41~~
26.1.10 TP to SQ: Texture stall ...... 45~~42~~
26.1.11 SQ to SP: Texture stall ...... 45~~42~~
26.1.12 SQ to SP: GPR and auto counter ...... 45~~42~~
26.1.13 SQ to SPx: Instructions ...... 46~~43~~
26.1.14 SP to SQ: Constant address load/ Predicate Set ...... 47~~44~~
26.1.15 SQ to SPx: constant broadcast ...... 47~~44~~
26.1.16 SP0 to SQ: Kill vector load ...... 47~~44~~
26.1.17 SQ to CP: RBBM bus ...... 47~~44~~
26.1.18 CP to SQ: RBBM bus ...... 47~~44~~
27. EXAMPLES OF PROGRAM EXECUTIONS ...... 48~~45~~

27.1.1 Sequencer Control of a Vector of Vertices .... 48~~45~~
27.1.2 Sequencer Control of a Vector of Pixels .... 49~~46~~
27.1.3 Notes .... 50~~47~~
28. OPEN ISSUES .... 50~~47~~
~~**1. OVERVIEW .... 6**~~
~~1.1 Top Level Block Diagram .... 8~~
~~1.2 Data Flow graph (SP) .... 10~~
~~1.3 Control Graph .... 11~~
~~**2. INTERPOLATED DATA BUS .... 11**~~
~~**3. INSTRUCTION STORE .... 14**~~
~~**4. SEQUENCER INSTRUCTIONS .... 16**~~
~~**5. CONSTANT STORES .... 16**~~
~~5.1 Memory organizations .... 16~~
~~5.2 Management of the re-mapping tables .... 16~~
~~5.2.1 Dirty bits .... 18~~
~~5.2.2 Free List Block .... 18~~
~~5.2.3 De-allocate Block .... 19~~
~~5.2.4 Operation of Incremental model .... 19~~
~~5.3 Constant Store Indexing .... 19~~
~~5.4 Real Time Commands .... 20~~
~~5.5 Constant Waterfalling .... 20~~
~~**6. LOOPING AND BRANCHES .... 21**~~
~~6.1 The controlling state. .... 21~~
~~6.2 The Control Flow Program .... 21~~
~~6.3 Data dependant predicate instructions .... 23~~
~~6.4 HW Detection of PV,PS .... 24~~
~~6.5 Register file indexing .... 24~~
~~6.6 Predicated Instruction support for Texture clauses .... 24~~
~~6.7 Debugging the Shaders .... 25~~
~~6.7.1 Method 1: Debugging registers .... 25~~
~~6.7.2 Method 2: Exporting the values in the GPRs (12) .... 25~~
~~**7. PIXEL KILL MASK .... 25**~~
~~**8. MULTIPASS VERTEX SHADERS (HOS) .... 26**~~
~~**9. REGISTER FILE ALLOCATION .... 26**~~
~~**10. FETCH ARBITRATION .... 27**~~
~~**11. ALU ARBITRATION .... 27**~~
~~**12. HANDLING STALLS .... 28**~~
~~**13. CONTENT OF THE RESERVATION STATION FIFOS .... 28**~~
~~**14. THE OUTPUT FILE .... 28**~~
~~**15. IJ FORMAT .... 28**~~
~~15.1 Interpolation of constant attributes .... 29~~
~~**16. STAGING REGISTERS .... 29**~~
~~**17. THE PARAMETER CACHE .... 31**~~
~~**18. VERTEX POSITION EXPORTING .... 31**~~
~~**19. EXPORTING ARBITRATION .... 31**~~
~~**20. EXPORT TYPES .... 31**~~

~~20.1 Vertex Shading .... 31~~
~~20.2 Pixel Shading .... 32~~
~~**21. SPECIAL INTERPOLATION MODES .... 32**~~
~~21.1 Real time commands .... 32~~
~~21.2 Sprites/ XY screen coordinates/ FB information .... 32~~
~~21.3 Auto generated counters .... 33~~
~~21.3.1 Vertex shaders .... 33~~
~~21.3.2 Pixel shaders .... 33~~
~~**22. STATE MANAGEMENT .... 33**~~
~~22.1 Parameter cache synchronization .... 33~~
~~**23. XY ADDRESS IMPORTS .... 34**~~
~~23.1 Vertex indexes imports .... 34~~
~~**24. REGISTERS .... 34**~~
~~24.1 Control .... 34~~
~~24.2 Context .... 34~~
~~**25. DEBUG REGISTERS .... 35**~~
~~25.1 Context .... 35~~
~~**26. INTERFACES .... 35**~~
~~26.1 External Interfaces .... 35~~
~~26.1.1 SC to SQ : IJ Control bus .... 36~~
~~26.1.2 SQ to SP: Interpolator bus .... 36~~
~~26.1.3 SQ to SP: Parameter Cache Read control bus .... 36~~
~~26.1.4 SQ to SX: Parameter Cache Mux control Bus .... 37~~
~~26.1.5 SQ to SP: Staging Register Data .... 37~~
~~26.1.6 PA to SQ : Vertex interface .... 37~~
~~26.1.7 SQ to CP: State report .... 41~~
~~26.1.8 SQ to SX: Control bus .... 41~~
~~26.1.9 SX to SQ : Output file control .... 41~~
~~26.1.10 SQ to TP: Control bus .... 41~~
~~26.1.11 TP to SQ: Texture stall .... 42~~
~~26.1.12 SQ to SP: Texture stall .... 42~~
~~26.1.13 SQ to SP: GPR, Parameter cache control and auto counter .... 42~~
~~26.1.14 SQ to SPx: Instructions .... 43~~
~~26.1.15 SP to SQ: Constant address load .... 44~~
~~26.1.16 SQ to SPx: constant broadcast .... 44~~
~~26.1.17 SP0 to SQ: Kill vector load .... 44~~
~~26.1.18 SQ to CP: RBBM bus .... 44~~
~~26.1.19 CP to SQ: RBBM bus .... 44~~
~~**27. EXAMPLES OF PROGRAM EXECUTIONS .... 44**~~
~~27.1.1 Sequencer Control of a Vector of Vertices .... 44~~
~~27.1.2 Sequencer Control of a Vector of Pixels .... 46~~

~~27.1.3 Notes..46~~

~~**28. OPEN ISSUES...47**~~

Revision Changes:

Rev 0.1 (Laurent Lefebvre) Date: May 7, 2001	First draft.
Rev 0.2 (Laurent Lefebvre) Date : July 9, 2001	Changed the interfaces to reflect the changes in the SP. Added some details in the arbitration section.
Rev 0.3 (Laurent Lefebvre) Date : August 6, 2001	Reviewed the Sequencer spec after the meeting on August 3, 2001.
Rev 0.4 (Laurent Lefebvre) Date : August 24, 2001	Added the dynamic allocation method for register file and an example (written in part by Vic) of the flow of pixels/vertices in the sequencer.
Rev 0.5 (Laurent Lefebvre) Date : September 7, 2001	Added timing diagrams (Vic)
Rev 0.6 (Laurent Lefebvre) Date : September 24, 2001	Changed the spec to reflect the new R400 architecture. Added interfaces.
Rev 0.7 (Laurent Lefebvre) Date : October 5, 2001	Added constant store management, instruction store management, control flow management and data dependant predication.
Rev 0.8 (Laurent Lefebvre) Date : October 8, 2001	Changed the control flow method to be more flexible. Also updated the external interfaces.
Rev 0.9 (Laurent Lefebvre) Date : October 17, 2001	Incorporated changes made in the 10/18/01 control flow meeting. Added a NOP instruction, removed the conditional_execute_or_jump. Added debug registers.
Rev 1.0 (Laurent Lefebvre) Date : October 19, 2001	Refined interfaces to RB. Added state registers.
Rev 1.1 (Laurent Lefebvre) Date : October 26, 2001	Added SEQ→SP0 interfaces. Changed delta precision. Changed VGT→SP0 interface. Debug Methods added.
Rev 1.2 (Laurent Lefebvre) Date : November 16, 2001	Interfaces greatly refined. Cleaned up the spec.
Rev 1.3 (Laurent Lefebvre) Date : November 26, 2001	Added the different interpolation modes.
Rev 1.4 (Laurent Lefebvre) Date : December 6, 2001	Added the auto incrementing counters. Changed the VGT→SQ interface. Added content on constant management. Updated GPRs.
Rev 1.5 (Laurent Lefebvre) Date : December 11, 2001	Removed from the spec all interfaces that weren't directly tied to the SQ. Added explanations on constant management. Added PA→SQ synchronization fields and explanation.
Rev 1.6 (Laurent Lefebvre) Date : January 7, 2002	Added more details on the staging register. Added detail about the parameter caches. Changed the call instruction to a Conditionnal_call instruction. Added details on constant management and updated the diagram.
Rev 1.7 (Laurent Lefebvre) Date : February 4, 2002	Added Real Time parameter control in the SX interface. Updated the control flow section.
Rev 1.8 (Laurent Lefebvre) Date : March 4, 2002	New interfaces to the SX block. Added the end of clause modifier, removed the end of clause instructions.
Rev 1.9 (Laurent Lefebvre) Date :	Rearangement of the CF instruction bits in order to ensure byte alignment

1. Overview

The sequencer is based on the R300 design. It chooses two ALU clauses and a fetch clause to execute, and executes all of the instructions in a clause before looking for a new clause of the same type. Two ALU clauses are executed interleaved to hide the ALU latency. Each vector will have eight fetch and eight ALU clauses, but clauses do not need to contain instructions. A vector of pixels or vertices ping-pongs along the sequencer FIFO, bouncing from fetch reservation station to alu reservation station. A FIFO exists between each reservation stage, holding up vectors until the vector currently occupying a reservation station has left. A vector at a reservation station can be chosen to execute. The sequencer looks at all eight alu reservation stations to choose an alu clause to execute and all eight fetch stations to choose a fetch clause to execute. The arbitrator will give priority to clauses/reservation stations closer to the bottom of the pipeline. It will not execute an alu clause until the fetch fetches initiated by the previous fetch clause have completed. There are two separate sets of reservation stations, one for pixel vectors and one for vertices vectors. This way a pixel can pass a vertex and a vertex can pass a pixel.

To support the shader pipe the sequencer also contains the shader instruction cache, constant store, control flow constants and texture state. The four shader pipes also execute the same instruction thus there is only one sequencer for the whole chip.

The sequencer first arbitrates between vectors of 64 vertices that arrive directly from primitive assembly and vectors of 16 quads (64 pixels) that are generated in the scan converter.

The vertex or pixel program specifies how many GPRs it needs to execute. The sequencer will not start the next vector until the needed space is available in the GPRs.

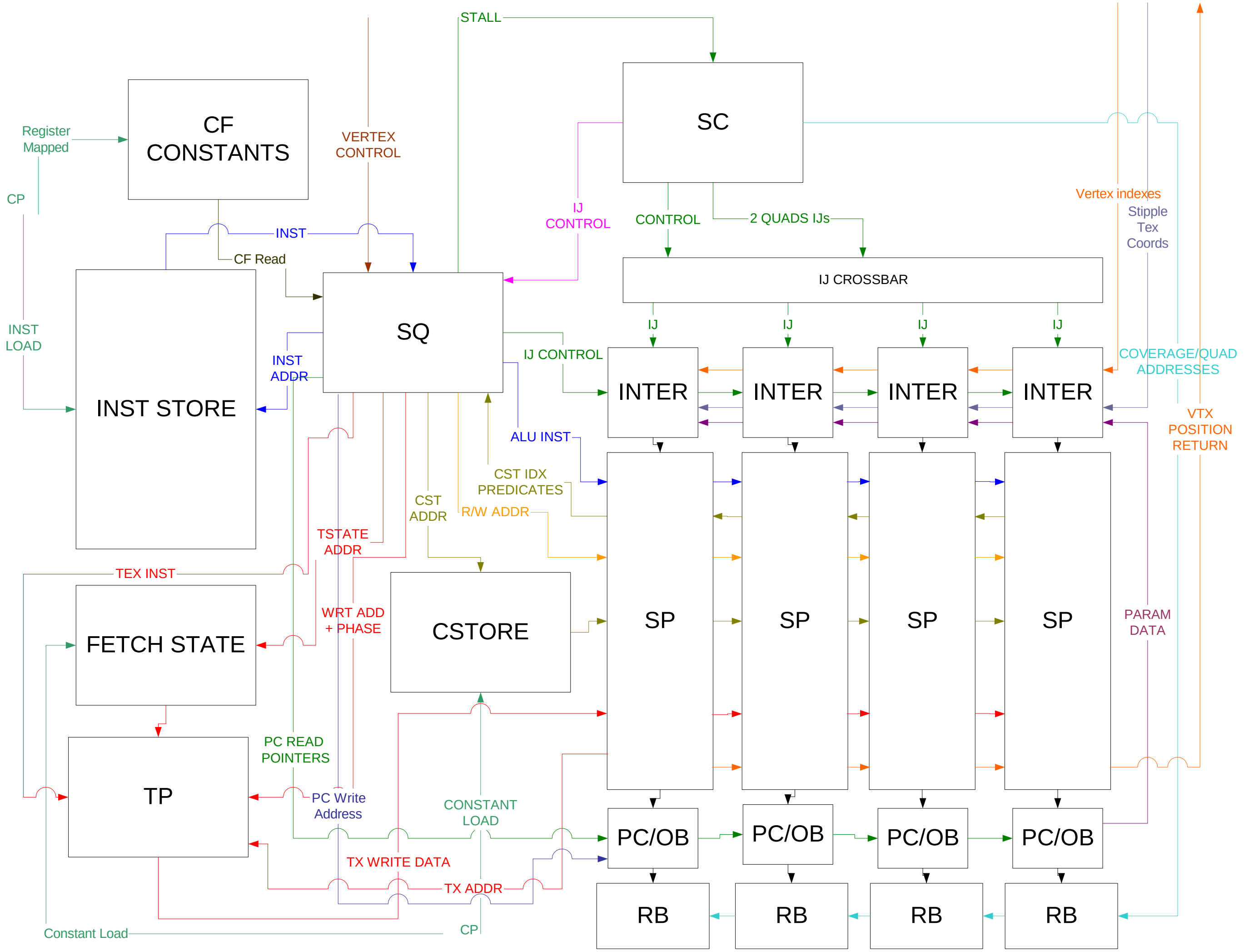


Figure 1: General Sequencer overview

1.1 Top Level Block Diagram

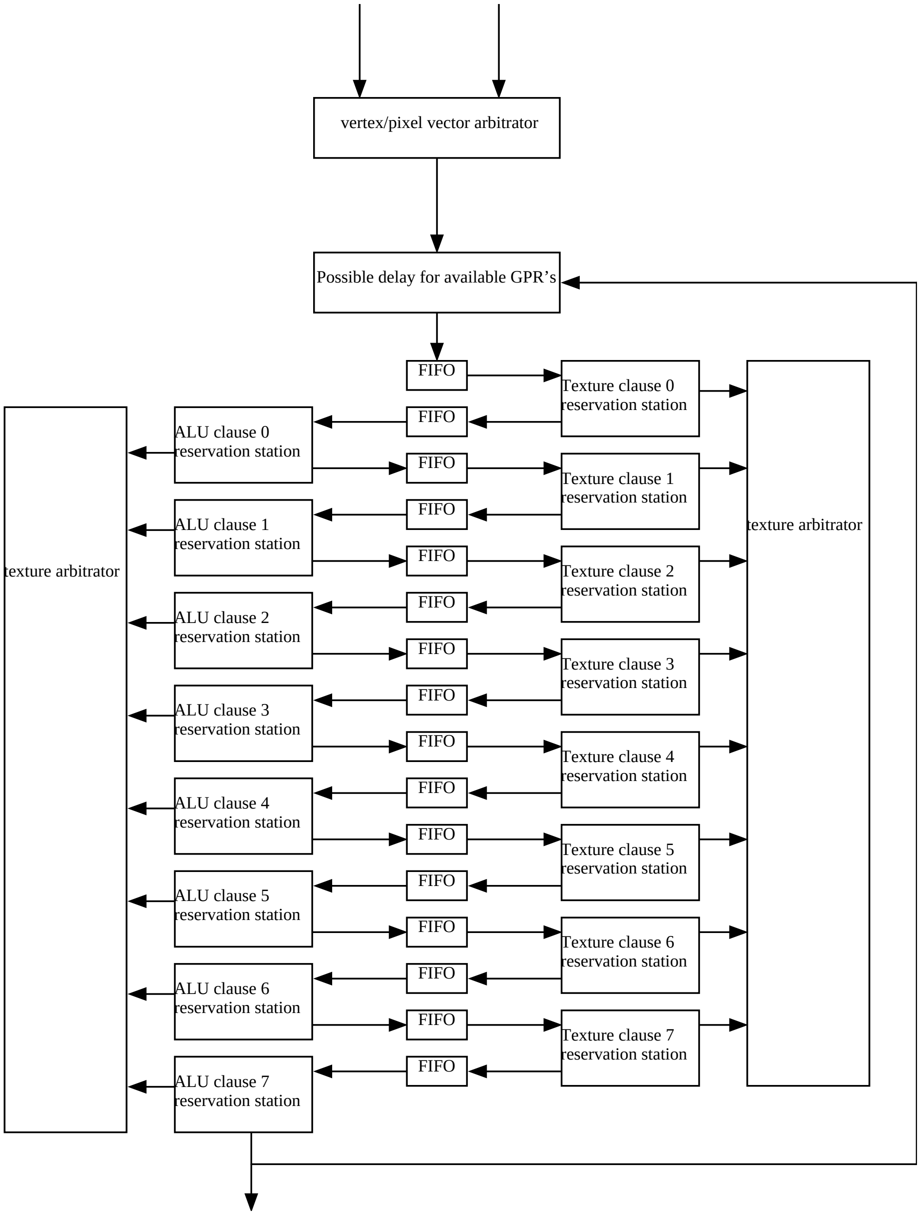

vertex/pixel vector arbitrator
Possible delay for available GPR's
FIFO
Texture clause 0 reservation station
Texture clause 1 reservation station
Texture clause 2 reservation station
Texture clause 3 reservation station
Texture clause 4 reservation station
Texture clause 5 reservation station
Texture clause 6 reservation station
Texture clause 7 reservation station
texture arbitrator
ALU clause 0 reservation station
ALU clause 1 reservation station
ALU clause 2 reservation station
ALU clause 3 reservation station
ALU clause 4 reservation station
ALU clause 5 reservation station
ALU clause 6 reservation station
ALU clause 7 reservation station
texture arbitrator

Figure 2: Reservation stations and arbiters

There are two sets of the above figure, one for vertices and one for pixels.

Depending on the arbitration state, the sequencer will either choose a vertex or a pixel packet. The control packet consists of 3 bits of state, 7 bits for the base address of the Shader program and some information on the coverage to determine fetch LOD plus other various small state bits.

On receipt of a packet, the input state machine (not pictured but just before the first FIFO) allocated enough space in the GPRs to store the interpolated values and temporaries. Following this, the barycentric coordinates (and XY screen position if needed) are sent to the interpolator, which will use them to interpolate the parameters and place the results into the GPRs. Then, the input state machine stacks the packet in the first FIFO.

On receipt of a command, the level 0 fetch machine issues a fetch request to the TP and corresponding GPR address for the fetch address (ta). A small command (tcmd) is passed to the fetch system identifying the current level number (0) as well as the GPR write address for the fetch return data. One fetch request is sent every 4 clocks causing the texturing of sixteen 2x2s worth of data (or 64 vertices). Once all the requests are sent the packet is put in FIFO 1.

Upon receipt of the return data, the fetch unit writes the data to the register file using the write address that was provided by the level 0 fetch machine and sends the clause number (0) to the level 0 fetch state machine to signify that the write is done and thus the data is ready. Then, the level 0 fetch machine increments the counter of FIFO 1 to signify to the ALU 0 that the data is ready to be processed.

On receipt of a command, the level 0 ALU machine first decrements the input FIFO 1 counter and then issues a complete set of level 0 shader instructions. For each instruction, the ALU state machine generates 3 source addresses, one destination address and an instruction. Once the last instruction has been issued, the packet is put into FIFO 2.

There will always be two active ALU clauses at any given time (and two arbiters). One arbiter will arbitrate over the odd instructions (4 clocks cycles) and the other one will arbitrate over the even instructions (4 clocks cycles). The only constraints between the two arbiters is that they are not allowed to pick the same clause number as the other one is currently working on if the packet is not of the same type (render state).

If the packet is a vertex packet, upon reaching ALU clause 3, it can export the position if the position is ready. So the arbiter must prevent ALU clause 3 to be selected if the positional buffer is full (or can't be accessed). Along with the positional data, if needed the sprite size and/or edge flags can also be sent.

A special case is for multipass vertex shaders, which can export 12 parameters per last 6 clauses to the output buffer. If the output buffer is full or doesn't have enough space the sequencer will prevent such a vertex group to enter an exporting clause.

Multipass pixel shaders can export 12 parameters to memory from the last clause only (7).

All other clauses process in the same way until the packet finally reaches the last ALU machine (7).

Only one pair of interleaved ALU state machines may have access to the register file address bus or the instruction decode bus at one time. Similarly, only one fetch state machine may have access to the register file address bus at one time. Arbitration is performed by three arbiter blocks (two for the ALU state machines and one for the fetch state machines). The arbiters always favor the higher number state machines, preventing a bunch of half finished jobs from clogging up the register files.

1.2 Data Flow graph (SP)

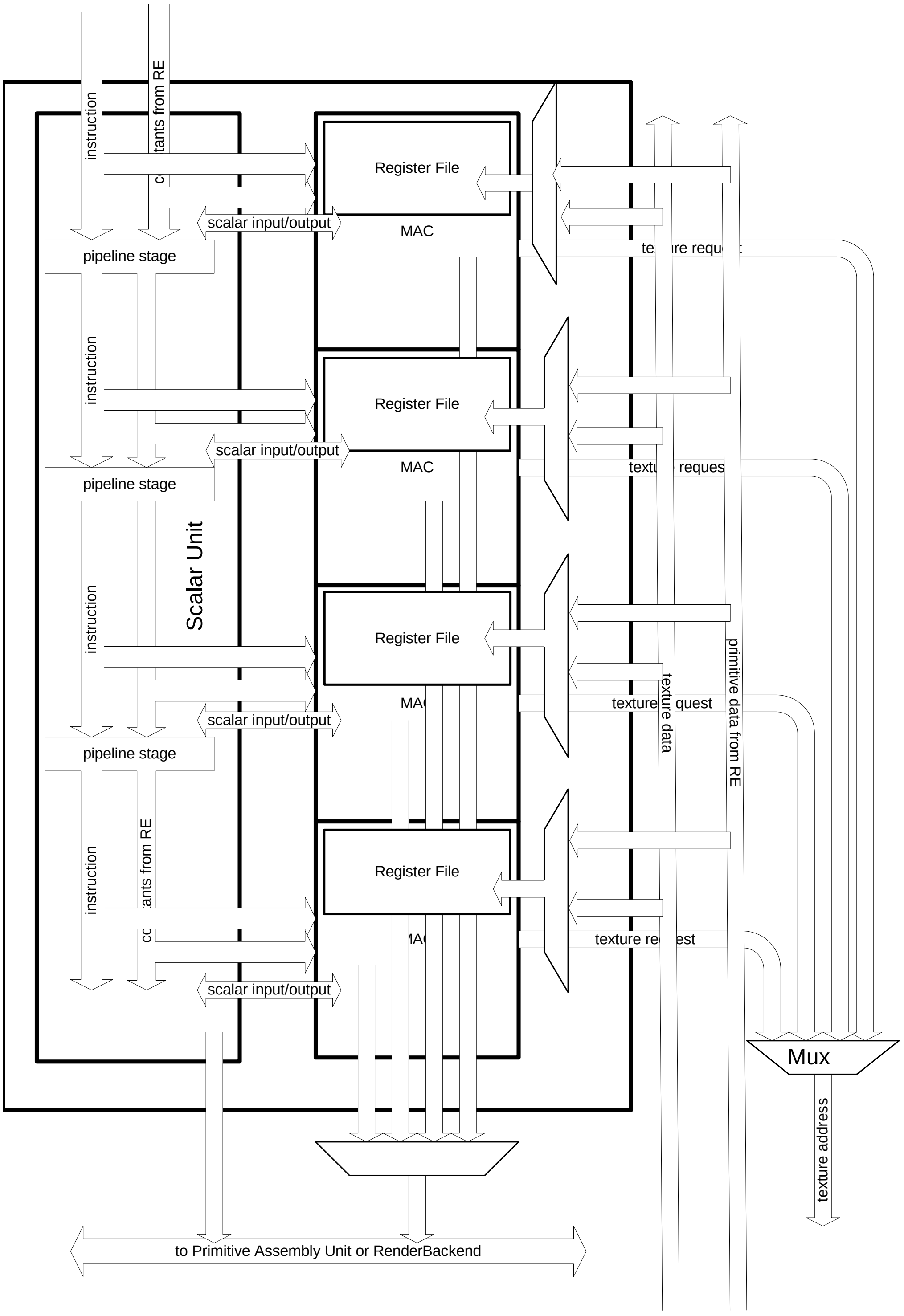


Figure 3: The shader Pipe

The gray area represents blocks that are replicated 4 times per shader pipe (16 times on the overall chip).

1.3 Control Graph

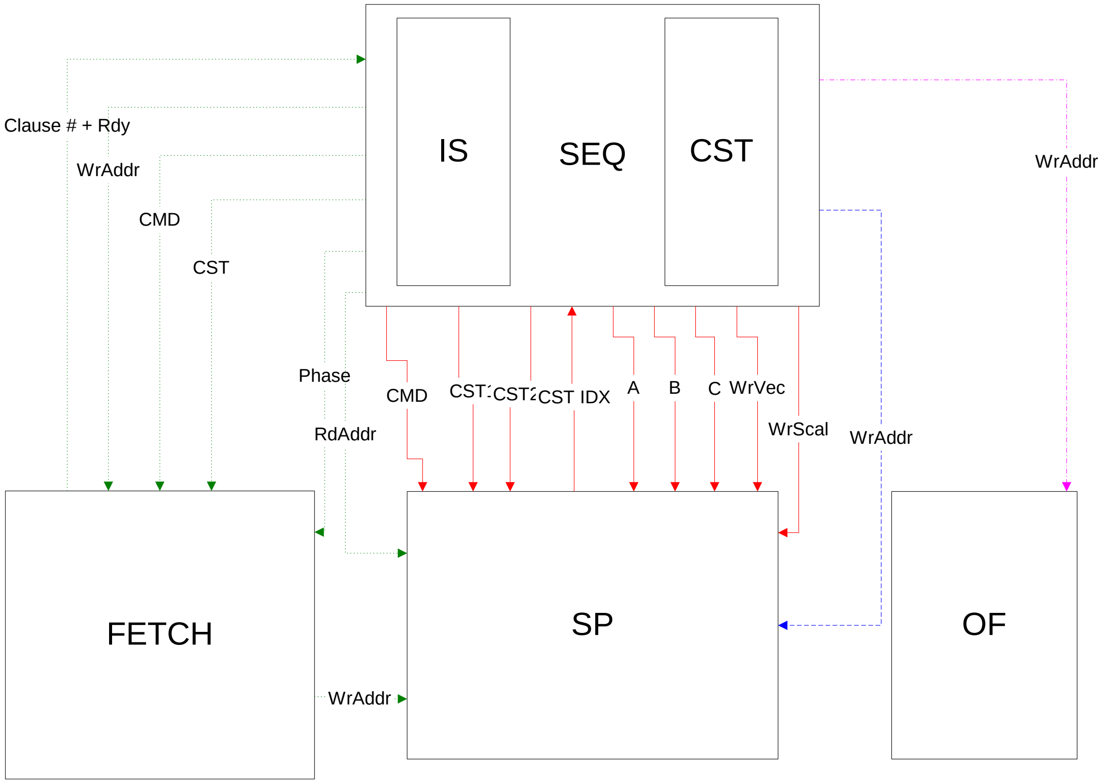


Figure 4: Sequencer Control interfaces

In green is represented the Fetch control interface, in red the ALU control interface, in blue the Interpolated/Vector control interface and in purple is the output file control interface.

2. Interpolated data bus

The interpolators contain an IJ buffer to pack the information as much as possible before writing it to the register file.

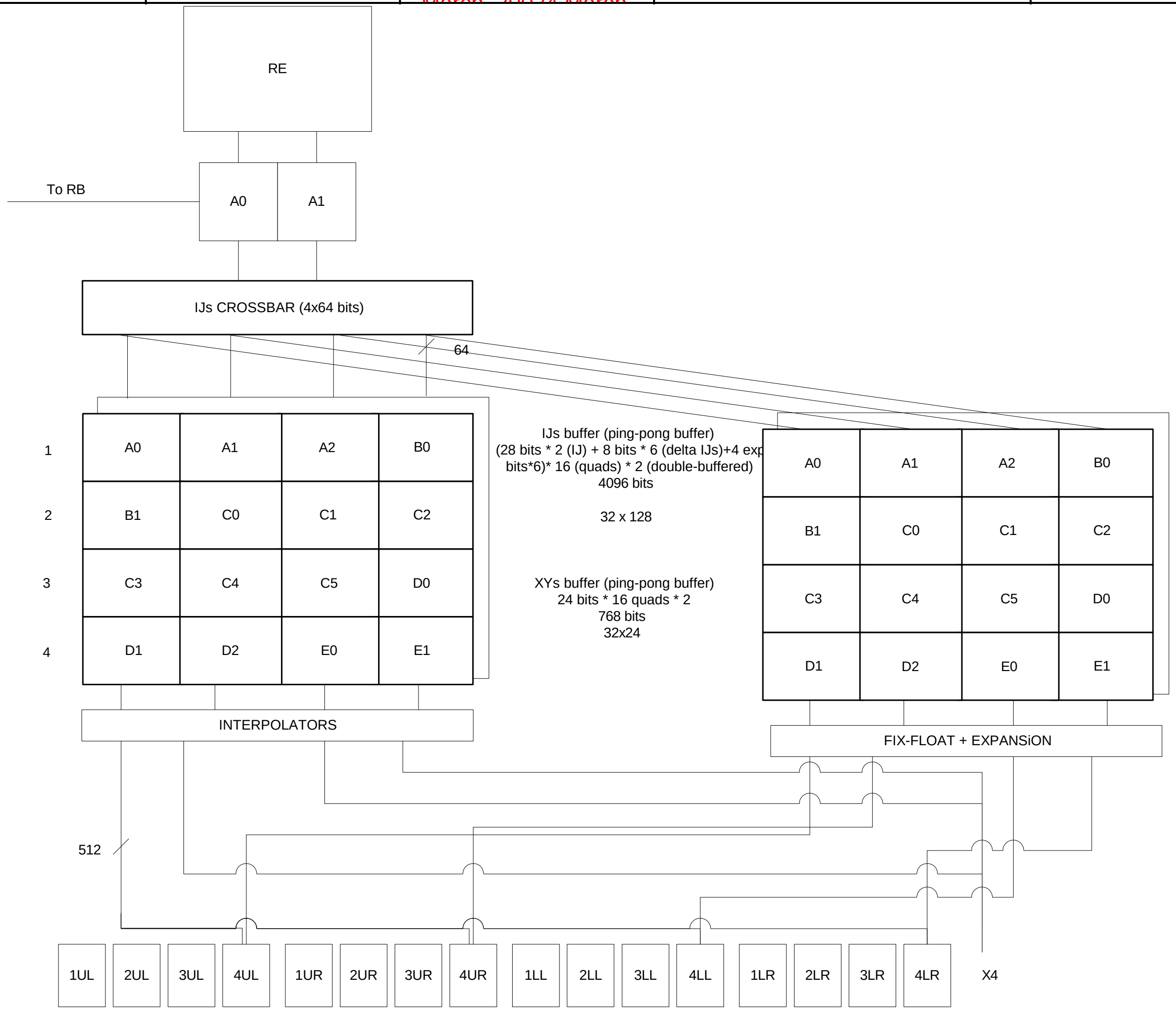


Figure 5: Interpolation buffers

WRITES

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21	T22	T23
SP 0	A0	A0	XY A0	B1	B1	XY B1	C3	C3	XY C3				D1	D1	XY D1									
SP 1	A1	A1	XY A1				C0	C0	XY C0	C4	C4	XY C4	D2	D2	XY D2									
SP 2	A2	A2	XY A2				C1	C1	XY C1	C5	C5	XY C5				E0	E0	XY E0						
SP 3				B0	B0	XY B0	C2	C2	XY C2				D0	D0	XY D0	E1	E1	XY E1						

READS

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21	T22	T23
SP 0	XY 0-3	XY 16-19	XY 32-35	XY 48-51	A0	B1	C3	D1					A0	B1	C3	D1					V 0-3	V 16-19	V 32-35	V 48-51
SP 1	XY 4-7	XY 20-23	XY 36-39	XY 52-55	A1		C4	D2		C0			A1		C4	D2		C0			V 4-7	V 20-23	V 36-39	V 52-55
SP 2	XY 8-11	XY 24-27	XY 40-43	XY 56-59	A2		C5			C1		E0	A2		C5			C1		E0	V 8-11	V 24-27	V 40-43	V 56-59
SP 3	XY 12-15	XY 28-31	XY 44-47	XY 60-63					B0	C2	D0	E1					B0	C2	D0	E1	V 12-15	V 28-31	V 44-47	V 60-63
		XY						P1								P2						VTX		

Figure 6: Interpolation timing diagram

Above is an example of a tile the sequencer might receive from the SC. The write side is how the data get stacked into the XY and IJ buffers, the read side is how the data is passed to the GPRs. The IJ information is packed in the IJ buffer 4 quads at a time or two clocks. The sequencer allows at any given time as many as four quads to interpolate a parameter. They all have to come from the same primitive. Then the sequencer controls the write mask to the GPRs to write the valid data in.

{ISSUE : Do we do the center + centroid approach using both IJ buffers?}

3. Instruction Store

There is going to be only one instruction store for the whole chip. It will contain 4096 instructions of 96 bits each.

It is likely to be a 1 port memory; we use 1 clock to load the ALU instruction, 1 clocks to load the Fetch instruction, 1 clock to load 2 control flow instructions and 1 clock to write instructions.

The instruction store is loaded by the CP thru the register mapped registers.

The next picture shows the various modes the CP can load the memory. The Sequencer has to keep track of the loading modes in order to wrap around the correct boundaries. The wrap-around points are arbitrary and they are specified in the VS_BASE and PIX_BASE control registers. The VS_BASE and PS_BASE context registers are used to specify for each context where its shader is in the instruction memory.

For the Real time commands the story is quite the same but for some small differences. There are no wrap-around points for real time so the driver must be careful not to overwrite regular shader data. The shared code (shared subroutines) uses the same path as real time.

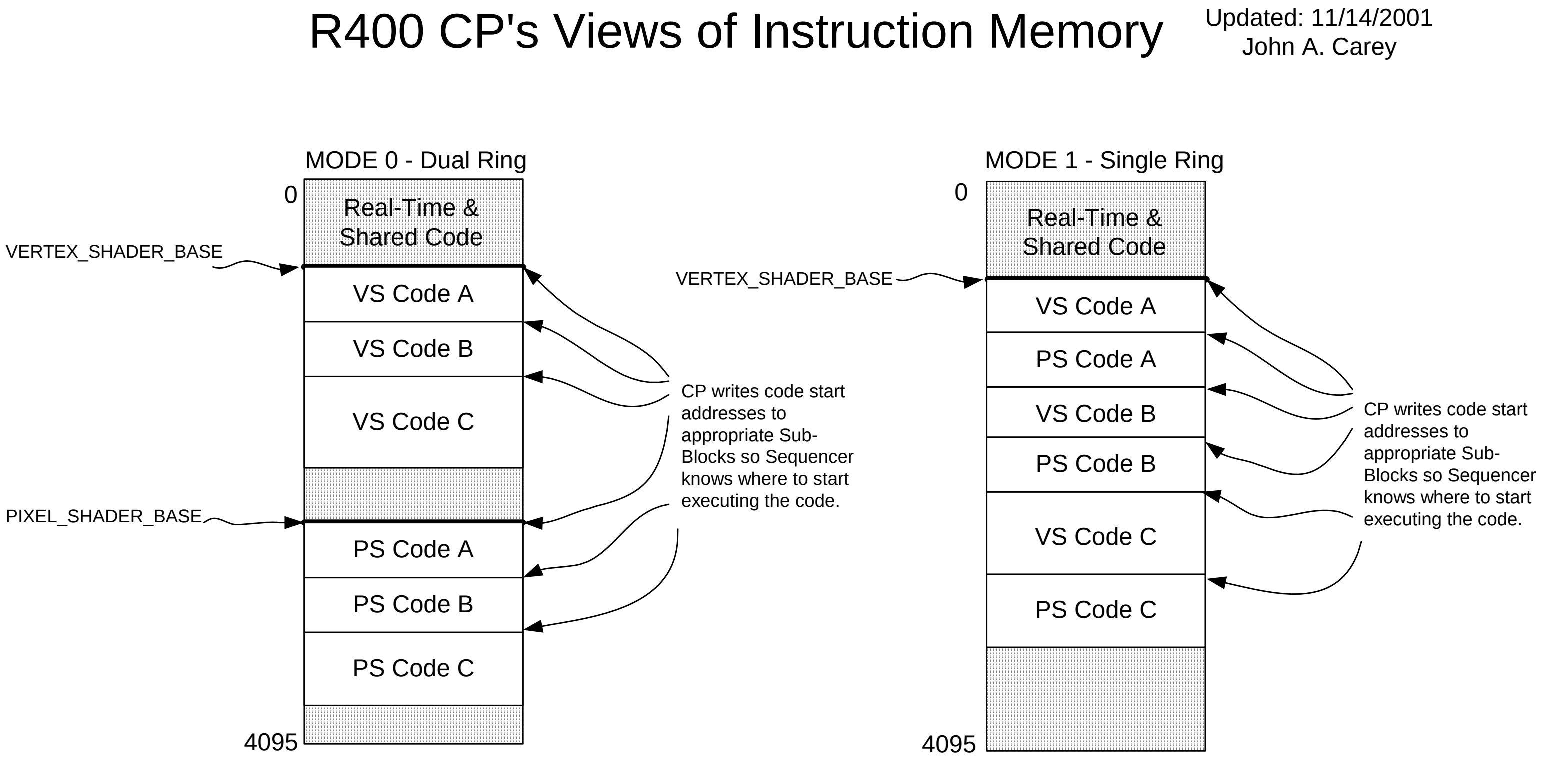


Figure 7: The CP's view of the instruction memory

4. Sequencer Instructions

All control flow instructions and move instructions are handled by the sequencer only. The ALUs will perform NOPs during this time (MOV PV,PV, PS,PS) if they have nothing else to do.

5. Constant Stores

5.1 Memory organizations

A likely size for the ALU constant store is 1024x128 bits. The read BW from the ALU constant store is 128 bits/clock and the write bandwidth is 32 bits/clock (directed by the CP bus size not by memory ports).

The maximum logical size of the constant store for a given shader is 256 constants. Or 512 for the pixel/vertex shader pair. The size of the re-mapping table is 128 lines (each line addresses 4 constants). The write granularity is 4 constants or 512 bits. It takes 16 clocks to write the four constants. Real time requires 256 lines in the physical memory (this is physically register mapped).

The texture state is also kept in a similar memory. The size of this memory is ~~128x192~~ 320x96 bits (128 texture states for regular mode, 32 states for RT). The memory thus holds 128 texture states (192 bits per state). The logical size exposes 32 different states total, which are going to be shared between the pixel and the vertex shader. The size of the re-mapping table to for the texture state memory is 32 lines (each line addresses 1 texture state lines in the real memory). The CP write granularity is 1 texture state lines (or 192 bits). The driver sends 512 bits but the CP ignores the top 320 bits. It thus takes 6 clocks to write the texture state. Real time requires 32 lines in the physical memory (this is physically register mapped).

The control flow constant memory doesn't sit behind a renaming table. It is register mapped and thus the driver must reload its content each time there is a change in the control flow constants. Its size is 320*32 because it must hold 8 copies of the 32 dwords of control flow constants and the loop construct constants must be aligned.

The constant re-mapping tables for texture state and ALU constants are logically register mapped for regular mode and physically register mapped for RT operation.

5.2 Management of the Control Flow Constants

The control flow constants are register mapped, thus the CP writes to the according register to set the constant, the SQ decodes the address and writes to the block pointed by its current base pointer (CF_WR_BASE). On the read side, one level of indirection is used. A register (SQ_CONTEXT_MISC.CF_RD_BASE) keeps the current base pointer to the control flow block. This register is copied whenever there is a state change. Should the CP write to CF after the state change, the base register is updated with the (current pointer number +1)% number of states. This way, if the CP doesn't write to CF the state is going to use the previous CF constants.

5.3 Management of the re-mapping tables

5.3.1 *R400 Constant management*

The sequencer is responsible to manage two re-mapping tables (one for the constant store and one for the texture state). On a state change (by the driver), the sequencer will broadside copy the contents of its re-mapping tables to a new one. We have 8 different re-mapping tables we can use concurrently.

The constant memory update will be incremental, the driver only need to update the constants that actually changed between the two state changes.

For this model to work in its simplest form, the requirement is that the physical memory MUST be at least twice as large as the logical address space + the space allocated for Real Time. In our case, since the logical address space

is 512 and the reserved RT space can be up to 256 entries, the memory must be of sizes 1280 and above. Similarly the size of the texture store must be of 32*2+32 = 96 entries and above.

5.3.2 *Proposal for R400LE constant management*

To make this scheme work with only 512+256 = 768 entries, upon reception of a CONTROL packet of state + 1, the sequencer would check for SQ_IDLE and PA_IDLE and if both are idle will erase the content of state to replace it with the new state (this is depicted in Figure 9: De-allocation mechanism~~Figure 9: De-allocation mechanism~~~~Figure 9: De-allocation mechanism~~). Note that in the case a state is cleared a value of 0 is written to the corresponding de-allocation counter location so that when the SQ is going to report a state change, nothing will be de-allocated upon the first report.

The second path sets all context dirty bits that were used in the current state to 1 (thus allowing the new state to reuse these physical addresses if needed).

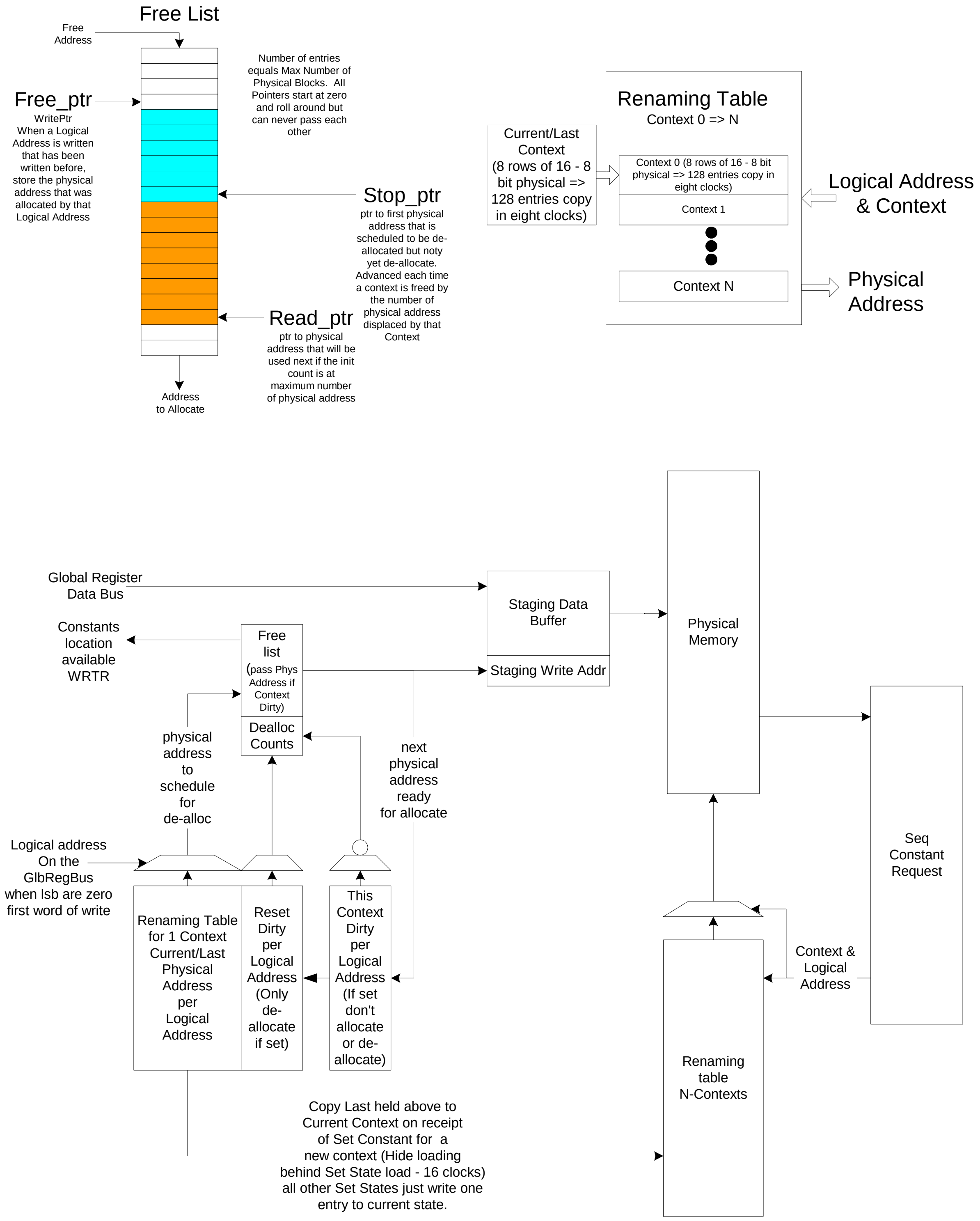

Free List
Free Address
Free_ptr
WritePtr
When a Logical Address is written that has been written before, store the physical address that was allocated by that Logical Address
Number of entries equals Max Number of Physical Blocks. All Pointers start at zero and roll around but can never pass each other
Stop_ptr
ptr to first physical address that is scheduled to be de-allocated but noty yet de-allocate. Advanced each time a context is freed by the number of physical address displaced by that Context
Read_ptr
ptr to physical address that will be used next if the init count is at maximum number of physical address
Address to Allocate
Renaming Table
Context 0 => N
Current/Last Context (8 rows of 16 - 8 bit physical => 128 entries copy in eight clocks)
Context 0 (8 rows of 16 - 8 bit physical => 128 entries copy in eight clocks)
Context 1
Context N
Logical Address & Context
Physical Address
Global Register Data Bus
Staging Data Buffer
Staging Write Addr
Physical Memory
Constants location available WRTR
Free list (pass Phys Address if Context Dirty)
Dealloc Counts
physical address to schedule for de-alloc
next physical address ready for allocate
Logical address On the GlbRegBus when lsb are zero first word of write
Renaming Table for 1 Context Current/Last Physical Address per Logical Address
Reset Dirty per Logical Address (Only de-allocate if set)
This Context Dirty per Logical Address (If set don't allocate or de-allocate)
Seq Constant Request
Context & Logical Address
Renaming table N-Contexts
Copy Last held above to Current Context on receipt of Set Constant for a new context (Hide loading behind Set State load - 16 clocks) all other Set States just write one entry to current state.

Figure 8: Constant management

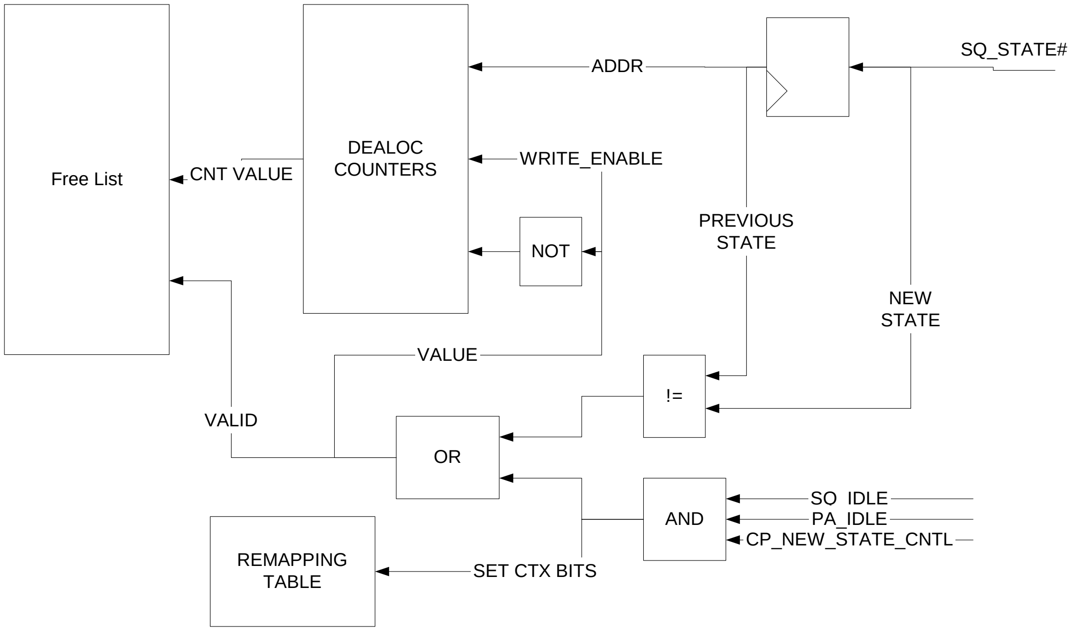


Figure 9: De-allocation mechanism for R400LE

5.3.3 *Dirty bits*

Two sets of dirty bits will be maintained per logical address. The first one will be set to zero on reset and set when the logical address is addressed. The second one will be set to zero whenever a new context is written and set for each address written while in this context. The reset dirty is not set, then writing to that logical address will not require de-allocation of whatever address stored in the renaming table. If it is set and the context dirty is not set, then the physical address store needs to be de-allocated and a new physical address is necessary to store the incoming data. If they are both set, then the data will be written into the physical address held in the renaming for the current logical address. No de-allocation or allocation takes place. This will happen when the driver does a set constant twice to the same logical address between context changes. NOTE: It is important to detect and prevent this, failure to do it will allow multiple writes to allocate all physical memory and thus hang because a context will not fit for rendering to start and thus free up space.

5.3.4 *Free List Block*

A free list block that would consist of a counter (called the IFC or Initial Free Counter) that would reset to zero and incremented every time a chunk of physical memory is used until they have all been used once. This counter would be checked each time a physical block is needed, and if the original ones have not been used up, us a new one, else check the free list for an available physical block address. The count is the physical address for when getting a chunk from the counter.

Storage of a free list big enough to store all physical block addresses.

Maintain three pointers for the free list that are reset to zero. The first one we will call write_ptr. This pointer will identify the next location to write the physical address of a block to be de-allocated. Note: we can never free more physical memory locations than we have. Once recording address the pointer will be incremented to walk the free list like a ring.

The second pointer will be called stop_ptr. The stop_ptr pointer will be advanced by the number of address chunks de-allocates when a context finishes. The address between the stop_ptr and write_ptr cannot be reused because they are still in use. But as soon as the context using then is dismissed the stop_ptr will be advanced.

The third pointer will be called read_ptr. This pointer will point will point to the next address that can be used for allocation as long as the read_ptr does not equal the stop_ptr and the IFC is at its maximum count.

5.3.5 *De-allocate Block*

This block will maintain a free physical address block count for each context. While in current context, a count shall be maintained specifying how many blocks were written into the free list at the write_ptr pointer. This count will be reset upon reset or when this context is active on the back and different than the previous context. It is actually a count of blocks in the previous context that will no longer be used. This count will be used to advance the write_ptr pointer to make available the set of physical blocks freed when the previous context was done. This allows the discard or de-allocation of any number of blocks in one clock.

5.3.6 *Operation of Incremental model*

The basic operation of the model would start with the write_ptr, stop_ptr, read_ptr pointers in the free list set to zero and the free list counter is set to zero. Also all the dirty bits and the previous context will be initialized to zero. When the first set constants happen, the reset dirty bit will not be set, so we will allocate a physical location from the free list counter because its not at the max value. The data will be written into physical address zero. Both the additional copy of the renaming table and the context zeros of the big renaming table will be updated for the logical address that was written by set start with physical address of 0. This process will be repeated for any logical address that are not dirty until the context changes. If a logical address is hit that has its dirty bits set while in the same context, both dirty bits would be set, so the new data will be over-written to the last physical address assigned for this logical address. When the first draw command of the context is detected, the previous context stored in the additional renaming table will be copied to the larger renaming table in the current (new) context location. Then the set constant logical address with be loaded with a new physical address during the copy and if the reset dirty was set, the physical address it replaced in the renaming table would be entered at the write_ptr pointer location on the free list and the write_ptr will be incremented. The de-allocation counter for the previous context (eight) will be incremented. This as set states come in for this context one of the following will happen:

1.) No dirty bits are set for the logical address being updated. A line will be allocated of the free-list counter or the free list at read_ptr pointer if read_ptr != to stop_ptr .
2.) Reset dirty set and Context dirty not set. A new physical address is allocated, the physical address in the renaming table is put on the free list at write_ptr and it is incremented along with the de-allocate counter for the last context.
3.) Context dirty is set then the data will be written into the physical address specified by the logical address.

This process will continue as long as set states arrive. This block will provide backpressure to the CP whenever he has not free list entries available (counter at max and stop_ptr == read_ptr). The command stream will keep a count of contexts of constants in use and prevent more than max constants contexts from being sent.

Whenever a draw packet arrives, the content of the re-mapping table is written to the correct re-mapping table for the context number. Also if the next context uses less constants than the current one all exceeding lines are moved to the free list to be de-allocated later. This happens in parallel with the writing of the re-mapping table to the correct memory.

Now preferable when the constant context leaves the last ALU clause it will be sent to this block and compared with the previous context that left. (Init to zero) If they differ than the older context will no longer be referenced and thus can be de-allocated in the physical memory. This is accomplished by adding the number of blocks freed this context to the stop_ptr pointer. This will make all the physical addresses used by this context available to the read_ptr allocate pointer for future allocation.

This device allows representation of multiple contexts of constants data with N copies of the logical address space. It also allows the second context to be represented as the first set plus some new additional data by just storing the delta's. It allows memory to be efficiently used and when the constants updates are small it can store multiple context. However, if the updates are large, less contexts will be stored and potentially performance will be degraded. Although it will still perform as well as a ring could in this case.

5.4 Constant Store Indexing

In order to do constant store indexing, the sequencer must be loaded first with the indexes (that come from the GPRs). There are 144 wires from the exit of the SP to the sequencer (9 bits pointers x 16 vertexes/clock). Since the data must pass thru the Shader pipe for the float to fixed conversion, there is a latency of 4 clocks (1 instruction)

between the time the sequencer is loaded and the time one can index into the constant store. The assembly will look like this

```
MOVA  R1.X,R2.X        // Loads the sequencer with the content of R2.X, also copies the content of R2.X into R1.X
NOP                    // latency of the float to fixed conversion
ADD   R3,R4,C0[R2.X]// Uses the state from the sequencer to add R4 to C0[R2.X] into R3
```

Note that we don’t really care about what is in the brackets because we use the state from the MOVA instruction. R2.X is just written again for the sake of simplicity and coherency.

The storage needed in the sequencer in order to support this feature is 2*64*9 bits = 1152 bits.

5.5 Real Time Commands

The real time commands constants are written by the CP using the register mapped registers allocated for RT. It works is the same way than when dealing with regular constant loads BUT in this case the CP is not sending a logical address but rather a physical address and the reads are not passing thru the re-mapping table but are directly read from the memory. The boundary between the two zones is defined by the CONST_EO_RT control register. Similarly, for the fetch state, the boundary between the two zones is defined by the TSTATE_EO_RT control register.

5.6 Constant Waterfalling

In order to have a reasonable performance in the case of constant store indexing using the address register, we are going to have the possibility of using the physical memory port for read only. This way we can read 1 constant per clock and thus have a worst-case waterfall mode of 1 vertex per clock. There is a small synchronization issue related with this as we need for the SQ to make sure that the constants where actually written to memory (not only sent to the sequencer) before it can allow the first vector of pixels or vertices of the state to go thru the ALUs. To do so, the sequencer keeps 8 bits (one per render state) and sets the bits whenever the last render state is written to memory and clears the bit whenever a state is freed.

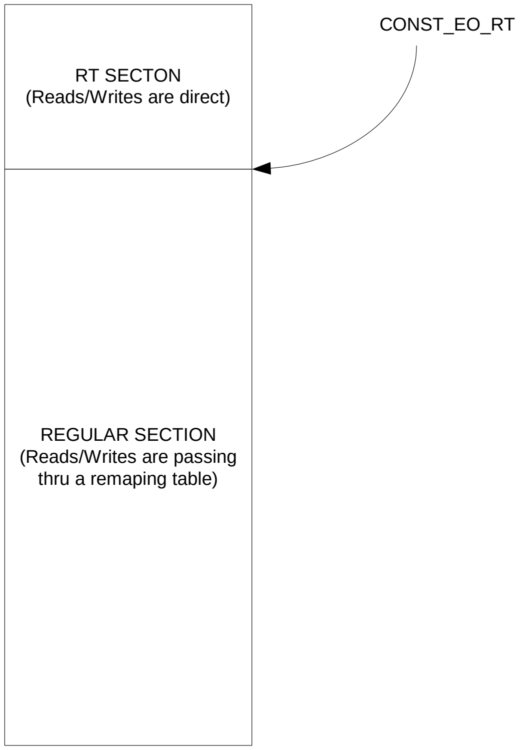


Figure 10: The instruction store

6. Looping and Branches

Loops and branches are planned to be supported and will have to be dealt with at the sequencer level. We plan on supporting constant loops and branches using a control program.

6.1 The controlling state.

The R400 controling state consists of:

Boolean[256:0]
Loop_count[7:0][31:0]
Loop_Start[7:0][31:0]
Loop_Step[7:0][31:0]

That is 256 Booleans and 32 loops.

We have a stack of 4 elements for nested calls of subroutines and 4 loop counters to allow for nested loops.

This state is available on a per shader program basis.

6.2 The Control Flow Program

Examples of control flow programs are located in the R400 programming guide document.

The basic model is as follows:

The render state defined the clause boundaries:

Vertex_shader_fetch[7:0][7:0]	// eight 8 bit pointers to the location where each clauses control program is located
Vertex_shader_alu[7:0][7:0]	// eight 8 bit pointers to the location where each clauses control program is located
Pixel_shader_fetch[7:0][7:0]	// eight 8 bit pointers to the location where each clauses control program is located
Pixel_shader_alu[7:0][7:0]	// eight 8 bit pointers to the location where each clauses control program is located

A pointer value of FF means that the clause doesn't contain any instructions.

The control program for a given clause is executed to completion before moving to another clause, (with the exception of the pick two nature of the alu execution). The control program is the only program aware of the clause boundaries.

The control program has nine basic instructions:

Execute
Conditional_execute
Conditional_Execute_Predicates
Conditional_jump
Conditionnal_Call
Return
Loop_start
Loop_end
NOP

Execute, causes the specified number of instructions in instruction store to be executed.
Conditional_execute checks a condition first, and if true, causes the specified number of instructions in instruction store to be executed.
Loop_start resets the corresponding loop counter to the start value on the first pass after it checks for the end condition and if met jumps over to a specified address.
Loop_end increments (decrements?) the loop counter and jumps back the specified number of instructions.
Conditionnal_Call jumps to an address and pushes the IP counter on the stack if the condition is met. On the return instruction, the IP is popped from the stack.

Conditional_execute_Predicates executes a block of instructions if all bits in the predicate vectors meet the condition.
Conditional_jumps jumps to an address if the condition is met.
NOP is a regular NOP

NOTE THAT ALL JUMPS MUST JUMP TO EVEN CFP ADDRESSES since there are two control flow instructions per memory line. Thus the compiler must insert NOPs where needed to align the jumps on even CFP addresses.

Also if the jump is logically bigger than pshader_cntl_size (or vshader_cntl_size) we break the program (clause) and set the debug registers. If an execute or conditional_execute is lower than cntl_size or bigger than size we also break the program (clause) and set the debug registers.

We have to fit instructions into 48 bits in order to be able to put two control flow instruction per line in the instruction store.

A value of 1 in the Addressing means that the address specified in the Exec Address field (or in the jump address field) is an ABSOLUTE address. If the addressing field is cleared (should be the default) then the address is relative to the base of the current shader program.

Note that whenever a field is marked as RESERVED, it is assumed that all the bits of the field are cleared (0).

Execute					
47	46… 42	41	40 … 24	23 … 12	11 … 0
Addressing	00001	Last	RESERVED	Instruction count	Exec Address

Execute up to 4k instructions at the specified address in the instruction memory. If Last is set, this is the last group of instructions of the clause.

NOP			
47	46 … 42	41	40 … 0
Addressing	00010	Last	RESERVED

This is a regular NOP. If Last is set, this is the last instruction of the clause.

Conditional_Execute								
47	46 … 42	41	40	~~40~~ 39 … ~~33~~32	~~32~~31	~~31~~ 30 … 24	23 … 12	11 … 0
Addressing	00011	Last	RESERVED	Boolean address	Condition	RESERVED	Instruction count	Exec Address

If the specified Boolean (8 bits can address 256 Booleans) meets the specified condition then execute the specified instructions (up to 4k instructions). If Last is set, then if the condition is met, this is the last group of instructions to be executed in the clause. If the condition is not met, we go on to the next control flow instruction.

Conditional_Execute_Predicates								
47	46 … 42	41	~~40~~ 40 … ~~35~~34	~~34~~ 33 … ~~33~~32	~~32~~31	~~31~~ 30 … 24	23 … 12	11 … 0
Addressing	00100	Last	RESERVED	Predicate vector	Condition	RESERVED	Instruction count	Exec Address

Check the AND/OR of all current predicate bits. If AND/OR matches the condition execute the specified number of instructions. We need to AND/OR this with the kill mask in order not to consider the pixels that aren't valid. If Last is set, then if the condition is met, this is the last group of instructions to be executed in the clause. If the condition is not met, we go on to the next control flow instruction.

Loop_Start				
47	46 … 42	41 … 17	16 … 12	11 … 0

Addressing	00101	RESERVED	loop ID	Jump address

Loop Start. Compares the loop iterator with the end value. If loop condition not met jump to the address. Forward jump only. Also computes the index value. The loop id must match between the start to end, and also indicates which control flow constants should be used with the loop.

Loop_End				
47	46 … 42	41 … 17	16 … 12	11 … 0
Addressing	00110	RESERVED	loop ID	start address

Loop end. Increments the counter by one, compares the loop count with the end value. If loop condition met, continue, else, jump BACK to the start of the loop.

The way this is described does not prevent nested loops, and the inclusion of the loop id make this easy to do.

Conditionnal_Call						
47	46 … 42	41 … ~~35~~34	~~34~~ 33 … ~~33~~32	31~~2~~	~~31~~ 30 … 12	11 … 0
Addressing	00111	RESERVED	Predicate vector	Condition	RESERVED	Jump address

If the condition is met, jumps to the specified address and pushes the control flow program counter on the stack.

Return		
47	46 … 42	41 … 0
Addressing	01000	RESERVED

Pops the topmost address from the stack and jumps to that address. If nothing is on the stack, the program will just continue to the next instruction.

Conditionnal_Jump							
47	46 … 42	41 … 40~~1~~	~~40~~ 39 … ~~33~~32	~~32~~31	~~31~~30	~~30~~ 29 … 12	11 … 0
Addressing	01001	RESERVED	Boolean address	Condition	FW only	RESERVED	Jump address

If condition met, jumps to the address. FORWARD jump only allowed if bit 31 set. Bit 31 is only an optimization for the compiler and should NOT be exposed to the API.

To prevent infinite loops, we will keep 9 bits loop iterators instead of 8 (we are only able to loop 256 times). If the counter goes higher than 255 then the loop_end or the loop_start instruction is going to break the loop and set the debug GPRs.

6.3 Data dependant predicate instructions

Data dependant conditionals will be supported in the R400. The only way we plan to support those is by supporting three vector/scalar predicate operations of the form:

PRED_SETE_# - similar to SETE except that the result is 'exported' to the sequencer.
PRED_SETNE_# - similar to SETNE except that the result is 'exported' to the sequencer.
PRED_SETGT_# - similar to SETGT except that the result is 'exported' to the sequencer
PRED_SETGTE_# - similar to SETGTE except that the result is 'exported' to the sequencer

For the scalar operations only we will also support the two following instructions:
PRED_SETE0_# – SETE0

PRED_SETE1_# – SETE1

The export is a single bit - 1 or 0 that is sent using the same data path as the MOVA instruction. The sequencer will maintain 4 sets of 64 bit predicate vectors (in fact 8 sets because we interleave two programs but only 4 will be exposed) and use it to control the write masking. This predicate is not maintained across clause boundaries. The # sign is used to specify which predicate set you want to use 0 thru 3.

Then we have two conditional execute bits. The first bit is a conditional execute “on” bit and the second bit tells us if we execute on 1 or 0. For example, the instruction:

P0_ADD_# R0,R1,R2

Is only going to write the result of the ADD into those GPRs whose predicate bit is 0. Alternatively, P1_ADD_# would only write the results to the GPRs whose predicate bit is set. The use of the P0 or P1 without precharging the sequencer with a PRED instruction is undefined.

{Issue: do we have to have a NOP between PRED and the first instruction that uses a predicate?}

6.4 HW Detection of PV,PS

Because of the control program, the compiler cannot detect statically dependant instructions. In the case of non-masked writes and subsequent reads the sequencer will insert uses of PV,PS as needed. This will be done by comparing the read address and the write address of consecutive instructions. For masked writes, the sequencer will insert NOPs wherever there is a dependant read/write.

The sequencer will also have to insert NOPs between PRED_SET and MOVA instructions and their uses.

6.5 Register file indexing

Because we can have loops in fetch clause, we need to be able to index into the register file in order to retrieve the data created in a fetch clause loop and use it into an ALU clause. The instruction will include the base address for register indexing and the instruction will contain these controls:

Bit7	Bit 6	
0	0	'absolute register'
0	1	'relative register'
1	0	'previous vector'
1	1	'previous scalar'

In the case of an absolute register we just take the address as is. In the case of a relative register read we take the base address and we add to it the loop_index and this becomes our new address that we give to the shader pipe.

The sequencer is going to keep a loop index computed as such:

Index = Loop_iterator*Loop_step + Loop_start.

We loop until loop_iterator = loop_count. Loop_step is a signed value [-128…127]. The computed index value is a 10 bit counter that is also signed. Its real range is [-256,256]. The tenth bit is only there so that we can provide an out of range value to the “indexing logic” so that it knows when the provided index is out of range and thus can make the necessary arrangements.

6.6 Predicated Instruction support for Texture clauses

For texture clauses, we support the following optimization: we keep 1 bit (thus 4 bits for the four predicate vectors) per predicate vector in the reservation stations. A value of 1 means that one ore more elements in the vector have a value of one (thus we have to do the texture fetches for the whole vector). A value of 0 means that no elements in the vector have his predicate bit set and we can thus skip over the texture fetch. **We have to make sure the invalid pixels aren’t considered with this optimization.**

6.7 Debugging the Shaders

In order to be able to debug the pixel/vertex shaders efficiently, we provide 2 methods.

6.7.1 *Method 1: Debugging registers*

Current plans are to expose 2 debugging, or error notification, registers:
1. address register where the first error occurred
2. count of the number of errors

The sequencer will detect the following groups of errors:
- count overflow
- constant indexing overflow
- register indexing overflow

Compiler recognizable errors:
- jump errors
    relative jump address > size of the control flow program
- call stack
    call with stack full
    return with stack empty

A jump error will always cause the program to break. In this case, a break means that a clause will halt execution, but allowing further clauses to be executed.

With all the other errors, program can continue to run, potentially to worst-case limits. The program will only break if the DB_PROB_BREAK register is set.

If indexing outside of the constant or the register range, causing an overflow error, the hardware is specified to return the value with an index of 0. This could be exploited to generate error tokens, by reserving and initializing the 0th register (or constant) for errors.

{ISSUE : Interrupt to the driver or not?}

6.7.2 *Method 2: Exporting the values in the GPRs (12)*

The sequencer will have a count register and an address register for this mode and 3 bits per clause specifying the execution mode for each clause. The modes can be :

1) Normal
2) Debug Kill
3) Debug Addr + Count

Under the normal mode execution follows the normal course. Under the kill mode, all control flow instructions are executed but all normal shader instructions of the clause are replaced by NOPs. Only debug_export instructions of clause 7 will be executed under the debug kill setting. Under the other mode, normal execution is done until we reach an address specified by the address register and instruction count (useful for loops) specified by the count register. After we have hit the instruction n times (n=count) we switch the clause to the kill mode.

Under the debug mode (debug kill OR debug Addr + count), it is assumed that clause 7 is always exporting 12 debug vectors and that all other exports to the SX block (position, color, z, ect) will been turned off (changed into NOPs) by the sequencer (even if they occur before the address stated by the ADDR debug register).

7. Pixel Kill Mask

A vector of 64 bits is kept by the sequencer per group of pixels/vertices. Its purpose is to optimize the texture fetch requests and allow the shader pipe to kill pixels using the following instructions:

MASK_SETE
MASK_SETNE
MASK_SETGT

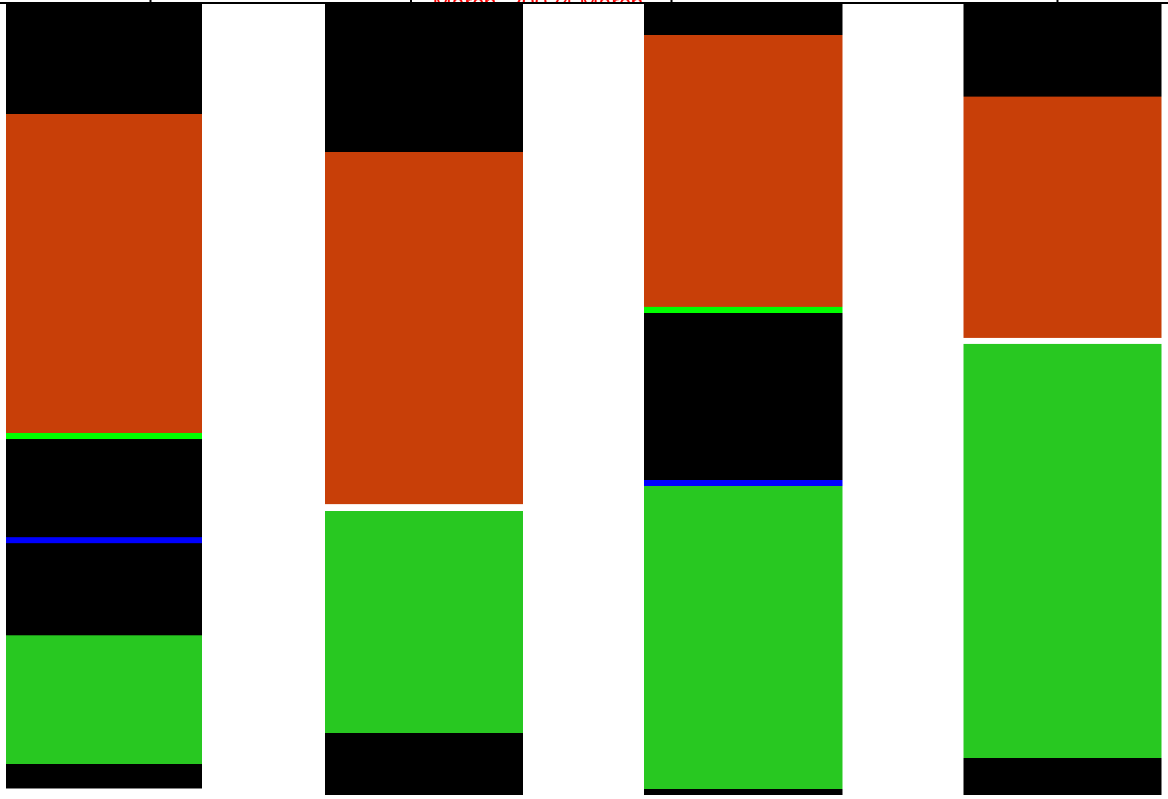
MASK_SETGTE

8. Multipass vertex shaders (HOS)

Multipass vertex shaders are able to export from the 6 last clauses but to memory ONLY.

9. Register file allocation

The register file allocation for vertices and pixels can either be static or dynamic. In both cases, the register file in managed using two round robins (one for pixels and one for vertices). In the dynamic case the boundary between pixels and vertices is allowed to move, in the static case it is fixed to 128-VERTEX_REG_SIZE for vertices and PIXEL_REG_SIZE for pixels.



Above is an example of how the algorithm works. Vertices come in from top to bottom; pixels come in from bottom to top. Vertices are in orange and pixels in green. The blue line is the tail of the vertices and the green line is the tail of the pixels. Thus anything between the two lines is shared. When pixels meets vertices the line turns white and the boundary is static until both vertices and pixels share the same "unallocated bubble". Then the boundary is allowed to move again. The numbering of the GPRs starts from the bottom of the picture at index 0 and goes up to the top at index 127.

10. Fetch Arbitration

The fetch arbitration logic chooses one of the 8 potentially pending fetch clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute. Once chosen, the clause state machine will send one 2x2 fetch per clock (or 4 fetches in one clock every 4 clocks) until all the fetch instructions of the clause are sent. This means that there cannot be any dependencies between two fetches of the same clause.

The arbitrator will not wait for the fetches to return prior to selecting another clause for execution. The fetch pipe will be able to handle up to X(?) in flight fetches and thus there can be a fair number of active clauses waiting for their fetch return data.

11. ALU Arbitration

ALU arbitration proceeds in almost the same way than fetch arbitration. The ALU arbitration logic chooses one of the 8 potentially pending ALU clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute. There are two ALU arbiters, one for the even clocks and one for the odd clocks. For example, here is the sequencing of two interleaved ALU clauses (E and O stands for Even and Odd sets of 4 clocks):

Einst0 Oinst0 Einst1 Oinst1 Einst2 Oinst2 Einst0 Oinst3 Einst1 Oinst4 Einst2 Oinst0…
Proceeding this way hides the latency of 8 clocks of the ALUs. Also note that the interleaving also occurs across clause boundaries.

12. Handling Stalls

When the output file is full, the sequencer prevents the ALU arbitration logic from selecting the last clause (this way nothing can exit the shader pipe until there is place in the output file. If the packet is a vertex packet and the position buffer is full (POS_FULL) then the sequencer also prevents a thread from entering the exporting clause (3?). The sequencer will set the OUT_FILE_FULL signal n clocks before the output file is actually full and thus the ALU arbiter will be able read this signal and act accordingly by not preventing exporting clauses to proceed.

13. Content of the reservation station FIFOs

The reservation FIFOs contain the state of the vector of pixels and vertices. We have two sets of those: one for pixels, and one for vertices. They contain 3 bits of Render State 7 bits for the base address of the GPRs, some bits for LOD correction and coverage mask information in order to fetch fetch for only valid pixels, the quad address.

14. The Output File

The output file is where pixels are put before they go to the RBs. The write BW to this store is 256 bits/clock. Just before this output file are staging registers with write BW 512 bits/clock and read BW 256 bits/clock. The staging registers are 4x128 (and there are 16 of those on the whole chip).

15. IJ Format

The IJ information sent by the PA is of this format on a per quad basis:

We have a vector of IJ's (one IJ per pixel at the centroid of the fragment or at the center of the pixel depending on the mode bit). The interpolation is done at a different precision across the 2x2. The upper left pixel's parameters are always interpolated at full 20x24 mantissa precision. Then the result of the interpolation along with the difference in IJ in reduced precision is used to interpolate the parameter for the other three pixels of the 2x2. Here is how we do it:

Assuming P0 is the interpolated parameter at Pixel 0 having the barycentric coordinates I(0), J(0) and so on for P1,P2 and P3. Also assuming that A is the parameter value at V0 (interpolated with I), B is the parameter value at V1 (interpolated with J) and C is the parameter value at V2 (interpolated with (1-I-J).

$$\Delta 01I = I(1) - I(0)$$
$$\Delta 01J = J(1) - J(0)$$
$$\Delta 02I = I(2) - I(0)$$
$$\Delta 02J = J(2) - J(0)$$
$$\Delta 03I = I(3) - I(0)$$
$$\Delta 03J = J(3) - J(0)$$

P0	P1
P2	P3

$$P0 = C + I(0) * (A - C) + J(0) * (B - C)$$
$$P1 = P0 + \Delta 01I * (A - C) + \Delta 01J * (B - C)$$
$$P2 = P0 + \Delta 02I * (A - C) + \Delta 02J * (B - C)$$
$$P3 = P0 + \Delta 03I * (A - C) + \Delta 03J * (B - C)$$

P0 is computed at 20x24 mantissa precision and P1 to P3 are computed at 8X24 mantissa precision. So far no visual degradation of the image was seen using this scheme.

Multiplies (Full Precision): 2
Multiplies (Reduced precision): 6
Subtracts 19x24 (Parameters): 2

Adds: 8

FORMAT OF P0's IJ : Mantissa 20 Exp 4 for I + Sign
Mantissa 20 Exp 4 for J + Sign

FORMAT of Deltas (x3):Mantissa 8 Exp 4 for I + Sign
Mantissa 8 Exp 4 for J + Sign

Total number of bits : 20*2 + 8*6 + 4*8 + 4*2 = 128

All numbers are kept using the un-normalized floating point convention: if exponent is different than 0 the number is normalized if not, then the number is un-normalized. The maximum range for the IJs (Full precision) is +/- 63 and the range for the Deltas is +/- 127.

15.1 Interpolation of constant attributes

Because of the floating point imprecision, we need to take special provisions if all the interpolated terms are the same or if two of the barycentric coordinates are the same.

We start with the premise that if A = B and B = C and C = A, then P0,1,2,3 = A. Since one or more of the IJ terms may be zero, so we extend this to:

```
if (A=B and B=C and C=A)
   P0,1,2,3 = A;
else if ((I = 0) or (J = 0)) and
          ((J = 0) or (1-I-J = 0)) and
          ((1-J-I = 0) or (I = 0))) {
                if(I != 0) {
                    P0 = A;
                } else if(J != 0) {
                    P0 = B;
                } else {
                    P0 = C;
               }
             //rest of the quad interpolated normally
}
else
{
             normal interpolation
}
```

16. Staging Registers

In order for the reuse of the vertices to be 14, the sequencer will have to re-order the data sent IN ORDER by the VGT for it to be aligned with the parameter cache memory arrangement. Given the following group of vertices sent by the VGT:

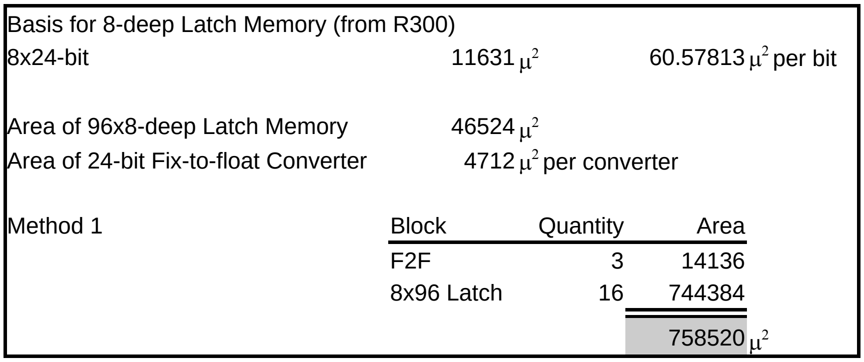
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 || 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 || 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 || 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63

The sequencer will re-arrange them in this fashion:

0 1 2 3 16 17 18 19 32 33 34 35 48 49 50 51 || 4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55 || 8 9 10 11 24 25 26 27 40 41 42 43 56 57 58 59 || 12 13 14 15 28 29 30 31 44 45 46 47 60 61 62 63

The || markers show the SP divisions. In the event a shader pipe is broken, the VGT will send padding to account for the missing pipe. For example, if SP1 is broken, vertices 4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55 will still be sent by the VGT to the SQ **BUT** will not be processed by the SP and thus should be considered invalid (by the SU and VGT).

The most straightforward, *non-compressed* interface method would be to convert, in the VGT, the data to 32-bit floating point prior to transmission to the VSISRs. In this scenario, the data would be transmitted to (and stored in) the VSISRs in full 32-bit floating point. This method requires three 24-bit fixed-to-float converters in the VGT. Unfortunately, it also requires and additional 3,072 bits of storage across the VSISRs. This interface is illustrated in Figure 12~~Figure 12~~~~Figure 12~~. The area of the fixed-to-float converters and the VSISRs for this method is roughly estimated as 0.759sqmm using the R300 process. The gate count estimate is shown in Figure 11~~Figure 11~~~~Figure 11~~.



Basis for 8-deep Latch Memory (from R300)				
8x24-bit	11631 μ^2		60.57813 μ^2 per bit	
Area of 96x8-deep Latch Memory	46524 μ^2			
Area of 24-bit Fix-to-float Converter	4712 μ^2 per converter			
Method 1	Block	Quantity	Area	
	F2F	3	14136	
	8x96 Latch	16	744384	
			758520	μ^2

Figure 11:Area Estimate for VGT to Shader Interface

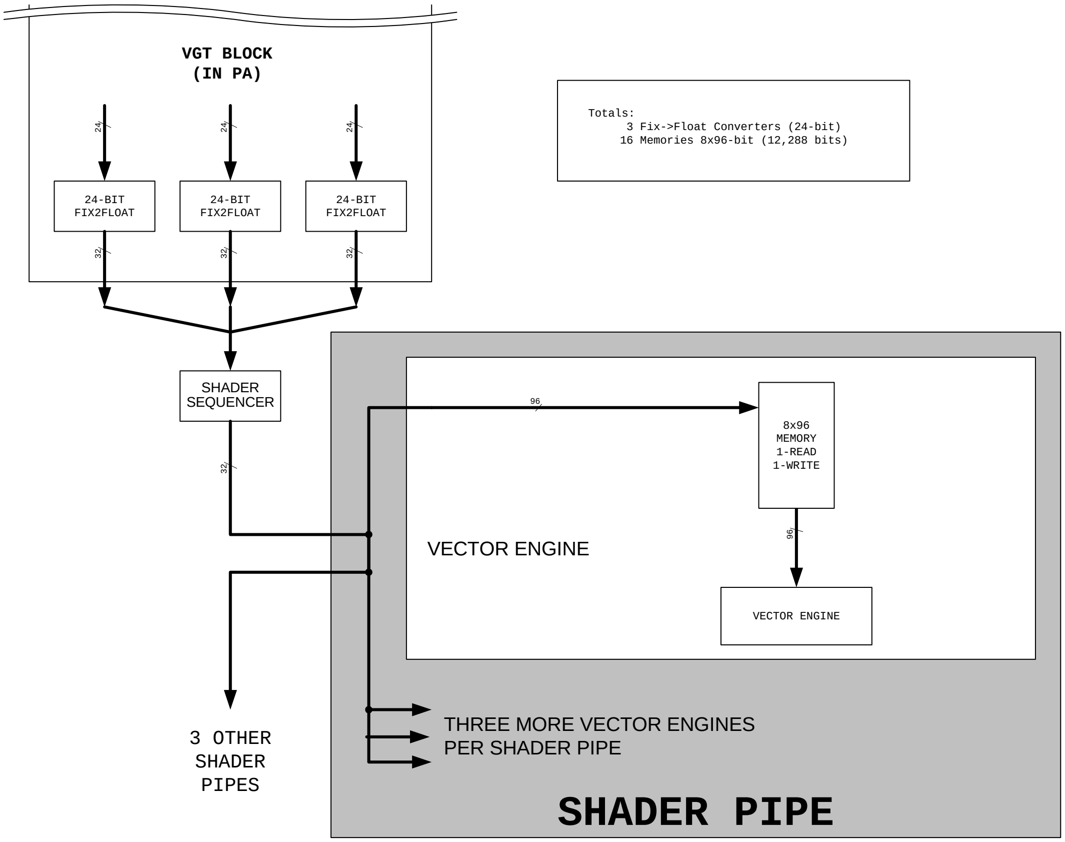


Figure 12:VGT to Shader Interface

17. The parameter cache

The parameter cache is where the vertex shaders export their data. It consists of 16 128x128 memories (1R/1W). The reuse engine will make it so that all vertexes of a given primitive will hit different memories. The allocation method for these memories is a simple round robin. The parameter cache pointers are mapped in the following way: 4MSBs are the memory number and the 7 LSBs are the address within this memory.

MEMORY NUMBER 4 bits	ADDRESS 7 bits

The PA generates the parameter cache addresses as the positions come from the SQ. All it needs to do is keep a Current_Location pointer (7 bits only) and as the positions comes increment the memory number. When the memory number field wraps around, the PA increments the Current_Location by VS_EXPORT_COUNT_7 (a snooped register from the SQ). As an example, say the memories are all empty to begin with and the vertex shader is exporting 8 parameters per vertex (VS_EXPORT_COUNT_7 = 8). The first position received is going to have the PC address 00000000000 the second one 00010000000, third one 00100000000 and so on up to 11110000000. Then the next position received (the 17th) is going to have the address 00000001000, the 18th 00010001000, the 19th 00100001000 and so on. The Current_location is NEVER reset BUT on chip resets. The only thing to be careful about is that if the SX doesn't send you a full group of positions (<64) then you need to fill the address space so that the next group starts correctly aligned (for example if you receive only 33 positions then you need to add 2*VS_EXPORT_COUNT_7to Current_Location and reset the memory count to 0 before the next vector begins).

18. Vertex position exporting

On clause 3 the vertex shader can export to the PA both the vertex position and the point sprite. It can also do so at clause 7 if not done at clause 3. The storage needed to perform the position export is at least 64x128 memories for the position and 64x32 memories for the sprite size. It is going to be taken in the pixel output fifo from the SX blocks. The clause where the position export occurs is specified by the EXPORT_LATE register. If turned on, it means that the export is going to occur at ALU clause 7 if unset position export occurs at clause 3.

19. Exporting Arbitration

Here are the rules for co-issuing exporting ALU clauses.

1) Position exports and position exports cannot be co-issued.

All other types of exports can be co-issued as long as there is place in the receiving buffer.

{ISSUE: Do we move the parameter caches to the SX?}

20. Export Types

The export type (or the location where the data should be put) is specified using the destination address field in the ALU instruction. Here is a list of all possible export modes:

20.1 Vertex Shading

0:15	- 16 parameter cache
16:31	- Empty (Reserved?)
32:43	- 12 vertex exports to the frame buffer and index
44:47	- Empty
48:59	- 12 debug export (interpret as normal vertex export)
60	- export addressing mode
61	- Empty
62	- position
63	- sprite size export that goes with position export (point_h,point_w,edgeflag,misc)

20.2 Pixel Shading

0	- Color for buffer 0 (primary)
1	- Color for buffer 1
2	- Color for buffer 2
3	- Color for buffer 3
4:7	- Empty
8	- Buffer 0 Color/Fog (primary)
9	- Buffer 1 Color/Fog
10	- Buffer 2 Color/Fog
11	- Buffer 3 Color/Fog
12:15	- Empty
16:31	- Empty (Reserved?)
32:43	- 12 exports for multipass pixel shaders.
44:47	- Empty
48:59	- 12 debug exports (interpret as normal pixel export)
60	- export addressing mode
61:62	- Empty
63	- Z for primary buffer (Z exported to 'alpha' component)

21. Special Interpolation modes

21.1 Real time commands

We are unable to use the parameter memory since there is no way for a command stream to write into it. Instead we need to add three 16x128 memories (one for each of three vertices x 16 interpolants). These will be mapped onto the register bus and written by type 0 packets, and output to the the parameter busses (the sequencer and/or PA need to be able to address the reatime parameter memory as well as the regular parameter store. For higher performance we should be able able to view them as two banks of 16 and do double buffering allowing one to be loaded, while the other is rasterized with. Most overlay shaders will need 2 or 4 scalar coordinates, one option might be to restrict the memory to 16x64 or 32x64 allowing only two interpolated scalars per cycle, the only problem I see with this is, if we view support for 16 vector-4 interpolants important (true only if we map Microsoft's high priority stream to the realtime stream), then the PA/sequencer need to support a realtime-specific mode where we need to address 32 vectors of parameters instead of 16. This mode is triggered by the primitive type: REAL TIME. The actual memories are in the in the SX blocks. The parameter data memories are hooked on the RBBM bus and are loaded by the CP using register mapped memory.

21.2 Sprites/ XY screen coordinates/ FB information

When working with sprites, one may want to overwrite the parameter 0 with SC generated data. Also, XY screen coordinates may be needed in the shader program. This functionality is controlled by the gen_I0 register (in SQ) in conjunction with the SND_XY register (in SC). Also it is possible to send the faceness information (for OGL front/back special operations) to the shader using the same control register. Here is a list of all the modes and how they interact together:

Gen_st is a bit taken from the interface between the SC and the SQ. This is the MSB of the primitive type. If the bit is set, it means we are dealing with Point AA, Line AA or sprite and in this case the vertex values are going to generated between 0 and 1.

Param_Gen_I0 disable, snd_xy disable, no gen_st – I0 = No modification
Param_Gen_I0 disable, snd_xy disable, gen_st – I0 = No modification
Param_Gen_I0 disable, snd_xy enable, no gen_st – I0 = No modification
Param_Gen_I0 disable, snd_xy enable, gen_st – I0 = No modification
Param_Gen_I0 enable, snd_xy disable, no gen_st – I0 = garbage, garbage, garbage, faceness
Param_Gen_I0 enable, snd_xy disable, gen_st – I0 = garbage, garbage, s, t
Param_Gen_I0 enable, snd_xy enable, no gen_st – I0 = screen x, screen y, garbage, faceness
Param_Gen_I0 enable, snd_xy enable, gen_st – I0 = screen x, screen y, s, t

21.3 Auto generated counters

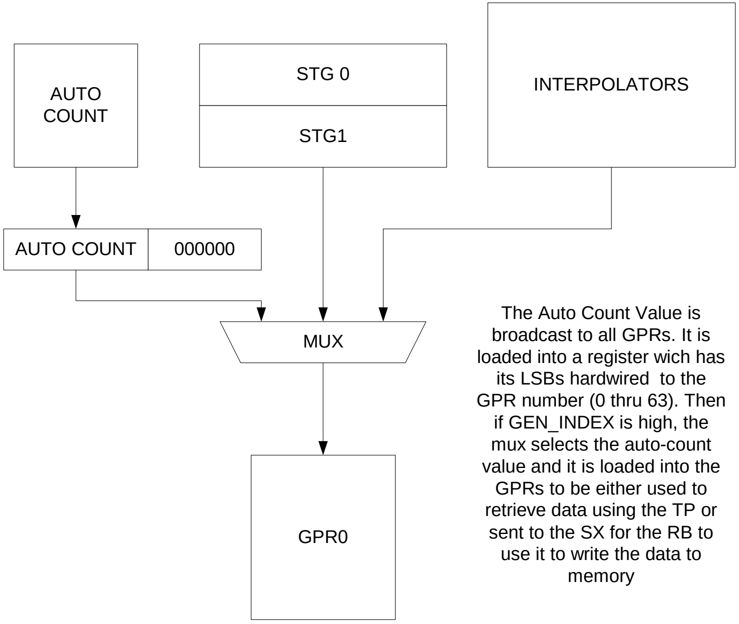
In the cases we are dealing with multipass shaders, the sequencer is going to generate a vector count to be able to both use this count to write the 1st pass data to memory and then use the count to retrieve the data on the 2nd pass. The count is always generated in the same way but it is passed to the shader in a slightly different way depending on the shader type (pixel or vertex). This is toggled on and off using the GEN_INDEX register. The sequencer is going to keep two counters, one for pixels and one for vertices. Every time a full vector of vertices or pixels is written to the GPRs the counter is incremented. Every time a state change is detected, the corresponding counter is reset. While there is only one count broadcast to the GPRs, the LSB are hardwired to specific values making the index different for all elements in the vector.

21.3.1 *Vertex shaders*

In the case of vertex shaders, if GEN_INDEX is set, the data will be put into the x field of the third register (it means that the compiler must allocate 3 GPRs in all multipass vertex shader modes).

21.3.2 *Pixel shaders*

In the case of pixel shaders, if GEN_INDEX is set and Param_Gen_I0 is enabled, the data will be put in the x field of the 2nd register (R1.x), else if GEN_INDEX is set the data will be put into the x field of the 1st register (R0.x).



The Auto Count Value is broadcast to all GPRs. It is loaded into a register wich has its LSBs hardwired to the GPR number (0 thru 63). Then if GEN_INDEX is high, the mux selects the auto-count value and it is loaded into the GPRs to be either used to retrieve data using the TP or sent to the SX for the RB to use it to write the data to memory

Figure 13: GPR input mux Control

22. State management

Every clock, the sequencer will report to the CP the oldest states still in the pipe. These are the states of the programs as they enter the last ALU clause.

22.1 Parameter cache synchronization

In order for the sequencer not to begin a group of pixels before the associated group of vertices has finished, the sequencer will keep a 6 bit count per state (for a total of 8 counters). These counters are initialized to 0 and every time a vertex shader exports its data TO THE PARAMETER CACHE, the corresponding pointer is incremented. When the SC sends a new vector of pixels with the SC_SQ_new_vector bit asserted, the sequencer will first check if the count is greater than 0 before accepting the transmission (it will in fact accept the transmission but then lower its ready to receive). Then the sequencer waits for the count to go to one and decrements it. The sequencer can then issue the group of pixels to the interpolators. Every time the state changes, the new state counter is initialized to 0.

23. XY Address imports

The SC will be able to send the XY addresses to the GPRs. It does so by interleaving the writes of the IJs (to the IJ buffer) with XY writes (to the XY buffer). Then when writing the data to the GPRs, the sequencer is going to interpolate the IJ data or pass the XY data thru a Fix→float converter and expander and write the converted values to the GPRs. The Xys are currently SCREEN SPACE COORDINATES. The values in the XY buffers will wrap. See section 21.2 for details on how to control the interpolation in this mode.

23.1 Vertex indexes imports

In order to import vertex indexes, we have 16 8x96 staging registers. These are loaded one line at a time by the VGT block (96 bits). They are loaded in floating point format and can be transferred in 4 or 8 clocks to the GPRs.

24. Registers

24.1 Control

REG_DYNAMIC	Dynamic allocation (pixel/vertex) of the register file on or off.
REG_SIZE_PIX	Size of the register file's pixel portion (minimal size when dynamic allocation turned on)
REG_SIZE_VTX	Size of the register file's vertex portion (minimal size when dynamic allocation turned on)
ARBITRATION_POLICY	policy of the arbitration between vertexes and pixels
INST_STORE_ALLOC	interleaved, separate
INST_BASE_VTX	start point for the vertex instruction store (RT always ends at vertex_base and Begins at 0)
INST_BASE_PIX	start point for the pixel shader instruction store
ONE_THREAD	debug state register. Only allows one program at a time into the GPRs
ONE_ALU	debug state register. Only allows one ALU program at a time to be executed (instead of 2)
INSTRUCTION	This is where the CP puts the base address of the instruction writes and type (auto-incremented on reads/writes) Register mapped
CONSTANTS	512*4 ALU constants + 32*6 Texture state 32 bits registers (logically mapped)
CONSTANTS_RT	256*4 ALU constants + 32*6 texture states? (physically mapped)
CONSTANT_EO_RT	This is the size of the space reserved for real time in the constant store (from 0 to CONSTANT_EO_RT). The re-mapping table operates on the rest of the memory
TSTATE_EO_RT	This is the size of the space reserved for real time in the fetch state store (from 0 to TSTATE_EO_RT). The re-mapping table operates on the rest of the memory
EXPORT_LATE	Controls whether or not we are exporting position from clause 3. If set, position exports occur at clause 7.

24.2 Context

VS_FETCH_{0…7}	eight 8 bit pointers to the location where each clauses control program is located
VS_ALU_{0…7}	eight 8 bit pointers to the location where each clauses control program is located
PS_FETCH_{0…7}	eight 8 bit pointers to the location where each clauses control program is located
PS_ALU_{0…7}	eight 8 bit pointers to the location where each clauses control program is located
PS_BASE	base pointer for the pixel shader in the instruction store
VS_BASE	base pointer for the vertex shader in the instruction store
VS_CF_SIZE	size of the vertex shader (# of instructions in control program/2)
PS_CF_SIZE	size of the pixel shader (# of instructions in control program/2)
PS_SIZE	size of the pixel shader (cntl+instructions)
VS_SIZE	size of the vertex shader (cntl+instructions)
PS_NUM_REG	number of GPRs to allocate for pixel shader programs
VS_NUM_REG	number of GPRs to allocate for vertex shader programs
PARAM_SHADE	One 16 bit register specifying which parameters are to be gouraud shaded (0 = flat, 1 = gouraud)
PROVO_VERT	0 : vertex 0, 1: vertex 1, 2: vertex 2, 3: Last vertex of the primitive
PARAM_WRAP	64 bits: for which parameters (and channels (xyzw)) do we do the cyl wrapping (0=linear, 1=cylindrical).
PS_EXPORT_MODE	0xxxx : Normal mode 1xxxx : Multipass mode If normal, bbbz where bbb is how many colors (0-4) and z is export z or not If multipass 1-12 exports for color.
~~VS_EXPORT_MASK~~	~~which of the last 6 ALU clauses is exporting (multipass only)~~
VS_EXPORT_MODE	0: position (1 vector), 1: position (2 vectors), 3:multipass
VS_EXPORT _COUNT_{0…6}	Six 4 bit counters representing the # of interpolated parameters exported in clause 7 (located in VS_EXPORT_COUNT_6) OR # of exported vectors to memory per clause in multipass mode (per clause)
PARAM_GEN_I0	Do we overwrite or not the parameter 0 with XY data and generated T and S values

GEN_INDEX	Auto generates an address from 0 to XX. Puts the results into R0-1 for pixel shaders and R2 for vertex shaders
CONST_BASE_VTX (9 bits)	Logical Base address for the constants of the Vertex shader
CONST_BASE_PIX (9 bits)	Logical Base address for the constants of the Pixel shader
CONST_SIZE_PIX (8 bits)	Size of the logical constant store for pixel shaders
CONST_SIZE_VTX (8 bits)	Size of the logical constant store for vertex shaders
INST_PRED_OPTIMIZE	Turns on the predicate bit optimization (if of, conditional_execute_predicates is always executed).
CF_BOOLEANS	256 boolean bits
CF_LOOP_COUNT	32x8 bit counters (number of times we traverse the loop)
CF_LOOP_START	32x8 bit counters (init value used in index computation)
CF_LOOP_STEP	32x8 bit counters (step value used in index computation)

25. DEBUG Registers

25.1 Context

DB_PROB_ADDR	instruction address where the first problem occurred
DB_PROB_COUNT	number of problems encountered during the execution of the program
DB_PROB_BREAK	break the clause if an error is found.
DB_INST_COUNT	instruction counter for debug method 2
DB_BREAK_ADDR	break address for method number 2
DB_CLAUSE_MODE_ALU_{0…7}	clause mode for debug method 2 (0: normal, 1: addr, 2: kill)
DB_CLAUSE_MODE_FETCH_{0…7}	clause mode for debug method 2 (0: normal, 1: addr, 2: kill)

25.2 Control

DB_ALUCST_MEMSIZE	Size of the physical ALU constant memory
DB_TSTATE_MEMSIZE	Size of the physical texture state memory

26. Interfaces

26.1 External Interfaces

Whenever an x is used, it means that the bus is broadcast to all units of the same name. For example, if a bus is named SQ→SPx it means that SQ is going to broadcast the same information to all SP instances.

26.1.1 *SC to SQ : IJ Control bus*

This is the control information sent to the sequencer in order to control the IJ fifos and all other information needed to execute a shader program on the sent pixels. This information is sent over 2 clocks, if SENDXY is asserted the next control packet is going to be ignored and XY information is going to be sent on the IJ bus (for the quads that where just sent). All pixels from the group of quads are from the same primitive, all quads of a vector are from the same render state.

Name	Direction	Bits	Description
SC_SQ_q_wr_mask	SC→SQ	4	Quad Write mask left to right
SC_SQ_lod_correct	SC→SQ	24	LOD correction per quad (6 bits per quad)
SC_SQ_param_ptr0	SC→SQ	11	P Store pointer for vertex 0
SC_SQ_param_ptr1	SC→SQ	11	P Store pointer for vertex 1
SC_SQ_param_ptr2	SC→SQ	11	P Store pointer for vertex 2
SC_SQ_end_of_vect	SC→SQ	1	End of the vector
SC_SQ_store_dealloc	SC→SQ	1	Deallocation token for the P Store
SC_SQ_state	SC→SQ	3	State/constant pointer
SC_SQ_valid_pixel	SC→SQ	16	Valid bits for all pixels
SC_SQ_null_prim	SC→SQ	1	Null Primitive (for PC deallocation purposes)
SC_SQ_end_of_prim	SC→SQ	1	End Of the primitive
SC_SQ_send_xy	SC→SQ	1	Sending XY information [XY information is going to be sent on the next clock]
SC_SQ_prim_type	SC→SQ	3	Real time command need to load tex cords from alternate buffer. Line AA, Point AA and Sprite reads their parameters from GEN_T and GEN_S GPRs. 000 : Normal 011 : Real Time 100 : Line AA 101 : Point AA 110 : Sprite
SC_SQ_new_vector	SC→SQ	1	This primitive comes from a new vector of vertices. Make sure that the corresponding vertex shader has finished before starting the group of pixels.
SC_SQ_RTRn	SQ→SC	1	Stalls the PA in n clocks
SC_SQ_RTS	SC→SQ	1	SC ready to send data

26.1.2 *SQ to SP: Interpolator bus*

Name	Direction	Bits	Description
SQ_SPx_interp_prim_type	SQ→SPx	3	Type of the primitive 000 : Normal 011 : Real Time 100 : Line AA 101 : Point AA 110 : Sprite
SQ_SPx_interp_ijline	SQ→SPx	2	Line in the IJ/XY buffer to use to interpolate
SQ_SPx_interp_mode	SQ→SPx	1	0: Use centroid buffer 1: Use center buffer
SQ_SPx_interp_buff_swap	SQ→SPx	1	Swap the IJ/XY buffers at the end of the interpolation
SQ_SPx_interp_gen_I0	SQ→SPx	1	Generate I0 or not. This tells the interpolators not to use the parameter cache but rather overwrite the data with interpolated 1 and 0. Overwrite if gen_I0 is high.

26.1.3 *SQ to SX: Interpolator bus*

Name	Direction	Bits	Description
SQ_SPx_interp_flat_vtx	SQ→SPx	2	Provoking vertex for flat shading
SQ_SPx_interp_flat_gouraud	SQ→SPx	1	Flat or gouraud shading
SQ_SPx_interp_cyl_wrap	SQ→SPx	4	Wich channel needs to be cylindrical wrapped
SQ_SXx_ptr1~~mux0~~	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_ptr2~~mux1~~	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_ptr3~~mux2~~	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_RT_switch	SQ→SXx	1	Selects between RT and Normal data
SQ_SXx_pc_wr_en	SQ→SXx	1	Write enable for the PC memories
SQ_SXx_pc_wr_addr	SQ→SXx	7	Write address for the PCs

26.1.4 SQ to SP: Staging Register Data

This is a broadcast bus that sends the VSISR information to the staging registers of the shader pipes.

Name	Direction	Bits	Description
SQ_SPx_vgt_vsisr_data	SQ→SPx	96	Pointers of indexes or HOS surface information
SQ_SPx_vgt_vsisr_double	SQ→SPx	1	0: Normal 96 bits per vert 1: double 192 bits per vert
SQ_SP0_data_valid	SQ→SP0	1	Data is valid
SQ_SP1_data_valid	SQ→SP1	1	Data is valid
SQ_SP2_data_valid	SQ→SP2	1	Data is valid
SQ_SP3_data_valid	SQ→SP3	1	Data is valid

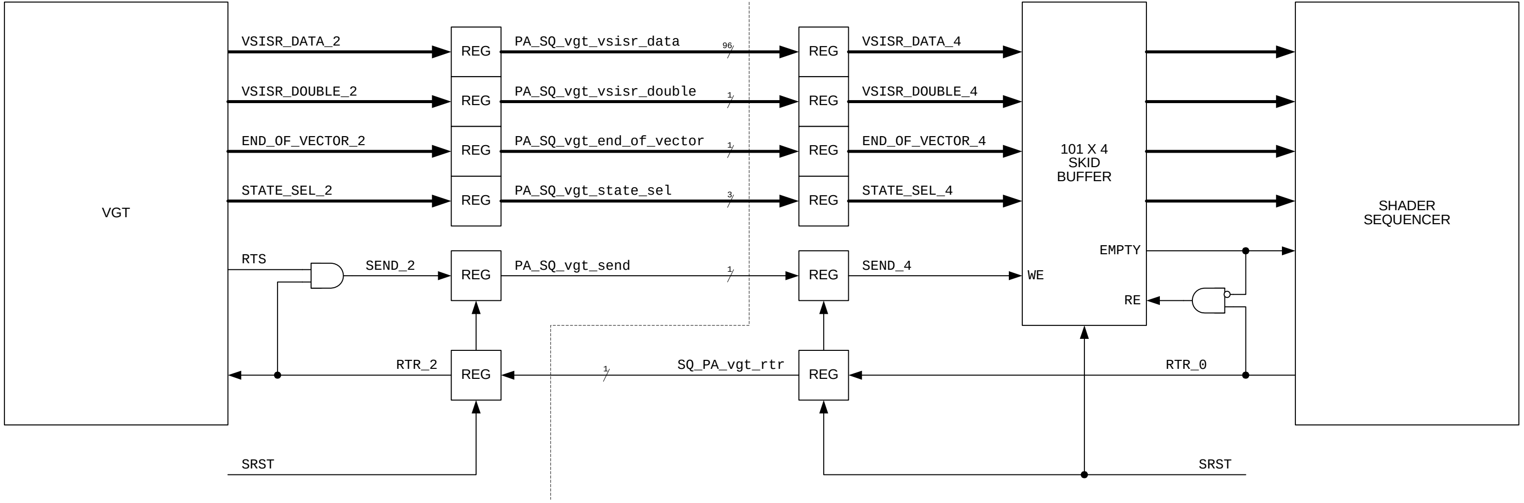
26.1.5 PA to SQ : Vertex interface

26.1.5.1 Interface Signal Table

The area difference between the two methods is not sufficient to warrant complicating the interface or the state requirements of the VSISRs. **Therefore, the POR for this interface is that the VGT will transmit the data to the VSISRs (via the Shader Sequencer) in full, 32-bit floating-point format.** The VGT can transmit up to six 32-bit floating-point values to each VSISR where four or more values require two transmission clocks. The data bus is 96 bits wide.

Name	Bits	Description
PA_SQ_vgt_vsisr_data	96	Pointers of indexes or HOS surface information
PA_SQ_vgt_vsisr_double	1	0: Normal 96 bits per vert 1: double 192 bits per vert
PA_SQ_vgt_end_of_vector	1	Indicates the last VSISR data set for the current process vector (for double vector data, "end_of_vector" is set on the second vector)
PA_SQ_vgt_vsisr_valid	1	Vsisr data is valid
PA_SQ_vgt_state	3	Render State (6*3+3 for constants). This signal is guaranteed to be correct when “PA_SQ_vgt_end_of_vector” is high.
PA_SQ_vgt_send	1	Data on the VGT_SQ is valid receive (see write-up for standard R400 SEND/RTR interface handshaking)
SQ_PA_vgt_rtr	1	Ready to receive (see write-up for standard R400 SEND/RTR interface handshaking)

26.1.5.2 Interface Diagrams


VGT
VSISR_DATA_2
VSISR_DOUBLE_2
END_OF_VECTOR_2
STATE_SEL_2
RTS
SEND_2
REG
PA_SQ_vgt_vsisr_data
96
PA_SQ_vgt_vsisr_double
1
PA_SQ_vgt_end_of_vector
1
PA_SQ_vgt_state_sel
3
PA_SQ_vgt_send
1
VSISR_DATA_4
VSISR_DOUBLE_4
END_OF_VECTOR_4
STATE_SEL_4
SEND_4
101 X 4
SKID
BUFFER
EMPTY
WE
RE
SHADER
SEQUENCER
RTR_2
1
SQ_PA_vgt_rtr
RTR_0
SRST
SRST

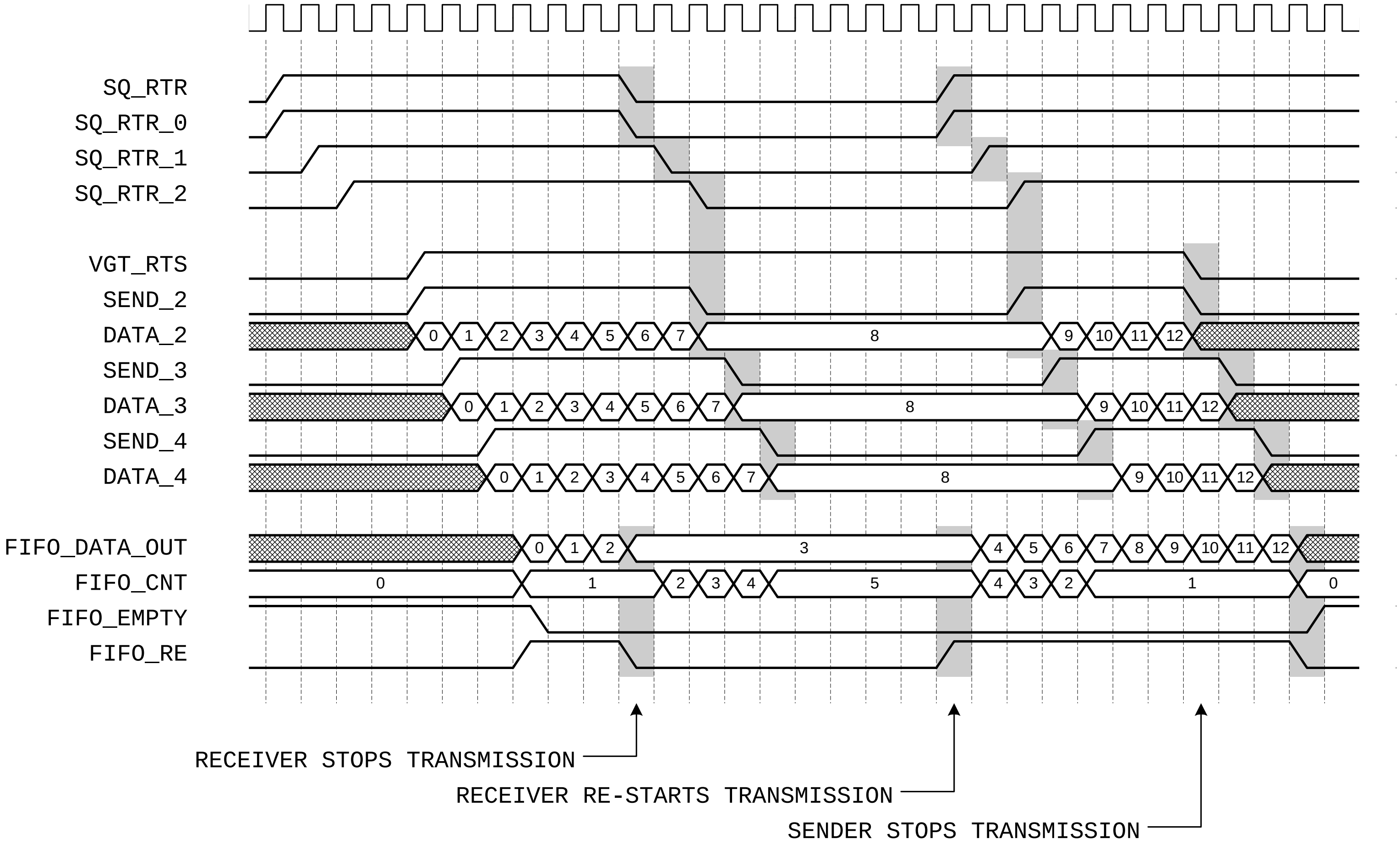


Figure 1. Detailed Logical Diagram for PA_SQ_vgt Interface.

26.1.6 SQ to CP: State report

Name	Direction	Bits	Description
SQ_CP_vrtx_ state	SEQ→CP	3	Oldest vertex state still in the pipe
SQ_CP_pix_state	SEQ→CP	3	Oldest pixel state still in the pipe

26.1.7 SQ to SX: Control bus

Name	Direction	Bits	Description
SQ_SXx_exp_Pixel	SQ→SXx	1	1: Pixel 0: Vertex
SQ_SXx_exp_Clause	SQ→SXx	3	Clause number, which is needed for vertex clauses
SQ_SXx_exp_State	SQ→SXx	3	State ID
SQ_SXx_exp_exportID	SQ→SXx	1	ALU ID

These fields are sent ~~synchronously with SP export data, described in SP0→SX0 interface~~every time the sequencer picks an exporting clause for execution.

26.1.8 SX to SQ : Output file control

Name	Direction	Bits	Description
SXx_SQ_Export_count_rdy	SXx→SQ	1	Raised by SX0 to indicate that the following two fields reflect the result of the most recent export
SXx_SQ_Export_Position	SXx→SQ	1	Specifies whether there is room for another position.
SXx_SQ_Export_Buffer	SXx→SQ	7	Specifies the space available in the output buffers. 0: buffers are full 1: 2K-bits available (32-bits for each of the 64 pixels in a clause) ... 64: 128K-bits available (16 128-bit entries for each of 64 pixels) 65-127: RESERVED

26.1.9 SQ to TP: Control bus

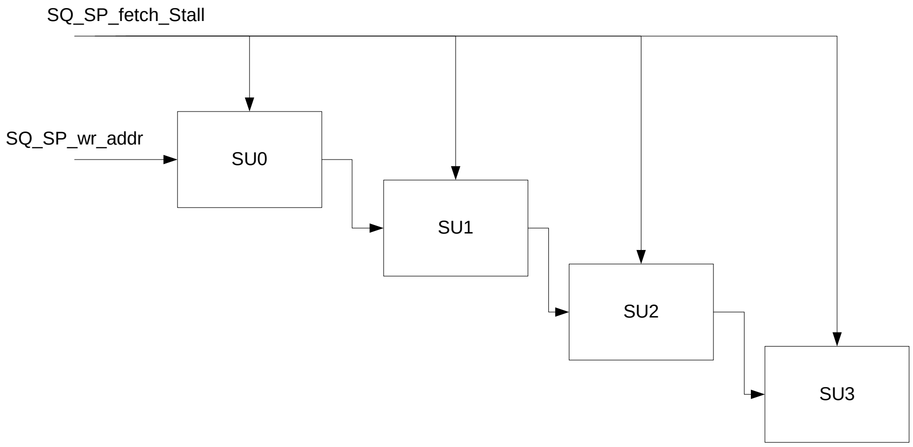
Once every clock, the fetch unit sends to the sequencer on which clause it is now working and if the data in the GPRs is ready or not. This way the sequencer can update the fetch counters for the reservation station fifos. The sequencer also provides the instruction and constants for the fetch to execute and the address in the register file where to write the fetch return data.

Name	Direction	Bits	Description
TPx_SQ_data_rdy	TPx→ SQ	1	Data ready
TPx_SQ_clause_num	TPx→ SQ	3	Clause number
TPx_SQ_Type	TPx→ SQ	1	Type of data sent (0:PIXEL, 1:VERTEX)
SQ_TPx_const	SQ→TPx	48	Fetch state sent over 4 clocks (192 bits total)
SQ_TPx_instuct	SQ→TPx	24	Fetch instruction sent over 4 clocks
SQ_TPx_end_of_clause	SQ→TPx	1	Last instruction of the clause
SQ_TPx_Type	SQ→TPx	1	Type of data sent (0:PIXEL, 1:VERTEX)
SQ_TPx_phase	SQ→TPx	2	Write phase signal
SQ_TP0_lod_correct	SQ→TP0	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP0_pmask	SQ→TP0	4	Pixel mask 1 bit per pixel
SQ_TP1_lod_correct	SQ→TP1	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP1_pmask	SQ→TP1	4	Pixel mask 1 bit per pixel
SQ_TP2_lod_correct	SQ→TP2	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP2_pmask	SQ→TP2	4	Pixel mask 1 bit per pixel
SQ_TP3_lod_correct	SQ→TP3	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP3_pmask	SQ→TP3	4	Pixel mask 1 bit per pixel

SQ_TPx_clause_num	SQ→TPx	3	Clause number
SQ_TPx_write_gpr_index	SQ->TPx	7	Index into Register file for write of returned Fetch Data

26.1.10 *TP to SQ: Texture stall*

The TP sends this signal to the SQ when its input buffer is full. The SQ is going to send it to the SP X clocks after reception (maximum of 3 clocks of pipeline delay).



Name	Direction	Bits	Description
TP_SQ_fetch_stall	TP→ SQ	1	Do not send more texture request if asserted

26.1.11 *SQ to SP: Texture stall*

Name	Direction	Bits	Description
SQ_SPx_fetch_stall	SQ→SPx	1	Do not send more texture request if asserted

26.1.12 *SQ to SP: GPR~~,~~ ~~Parameter cache control~~ and auto counter*

Name	Direction	Bits	Description
SQ_SPx_gpr_wr_addr	SQ→SPx	7	Write address
SQ_SPx_gpr_rd_addr	SQ→SPx	7	Read address
SQ_SPx_gpr_r~~e~~d_~~addr~~en	SQ→SPx	1	Read Enable
SQ_SPx_gpr_~~we~~wr_~~addr~~en	SQ→SPx	1	Write Enable for the GPRs
SQ_SPx_gpr_phase_mux	SQ→SPx	2	The phase mux (arbitrates between inputs, ALU SRC reads and writes)
SQ_SPx_channel_mask	SQ→SPx	4	The channel mask
SQ_SP0_pixel_mask	SQ→SP0	4	The pixel mask
SQ_SP1_pixel_mask	SQ→SP1	4	The pixel mask
SQ_SP2_pixel_mask	SQ→SP2	4	The pixel mask
SQ_SP3_pixel_mask	SQ→SP3	4	The pixel mask
SQ_SPx_gpr_input_mux	SQ→SPx	2	When the phase mux selects the inputs this tells from which source to read from: Interpolated data, VTX0, VTX1, autogen counter.
SQ_SPx_index_count	SQ→SPx	12?	Index count, common for all shader pipes

26.1.13 *SQ to SPx: Instructions*

Name	Direction	Bits	Description
SQ_SPx_instruct_start	SQ→SPx	1	Instruction start
SQ_SP_instruct	SQ→SPx	21	Transferred over 4 cycles 0: SRC A Select 2:0 SRC A Argument Modifier 3:3 SRC A swizzle 11:4 VectorDst 17:12 Unused 20:18 -- - 1: SRC B Select 2:0 SRC B Argument Modifier 3:3 SRC B swizzle 11:4 ScalarDst 17:12 Unused 20:18 -- - 2: SRC C Select 2:0 SRC C Argument Modifier 3:3 SRC C swizzle 11:4 Unused 20:12 -- - 3: Vector Opcode 4:0 Scalar Opcode 10:5 Vector Clamp 11:11 Scalar Clamp 12:12 Vector Write Mask 16:13 Scalar Write Mask 20:17
SQ_SPx_exp_exportID	SQ→SPx	1	ALU ID
SQ_SPx_stall	SQ→SPx	1	Stall signal
SQ_SPx_export_count	SQ→SPx	3	Each set of four pixels or vectors is exported over eight clocks. This field specifies where the SP is in that sequence.
SQ_SPx_export_last	SQ→SPx	1	Asserted on the first shader count of the last export of the clause
SQ_SP0_export_pvalid	SQ→SP0	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP0_export_wvalid	SQ→SP0	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors
SQ_SP1_ export_pvalid	SQ→SP1	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP1_ export_wvalid	SQ→SP1	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors
SQ_SP2_ export_pvalid	SQ→SP2	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP2_ export_wvalid	SQ→SP2	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors
SQ_SP3_ export_pvalid	SQ→SP3	4	Result of pixel kill in the shader pipe, which must be

			output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP3_ export_wvalid	SQ→SP3	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors

26.1.14 *SP to SQ: Constant address load/ Predicate Set*

Name	Direction	Bits	Description
SP0_SQ_const_addr	SP0→SQ	36	Constant address load / predicate vector load (4 bits only) to the sequencer
SP0_SQ_valid	SP0→SQ	1	Data valid
SP1_SQ_const_addr	SP1→SQ	36	Constant address load / predicate vector load (4 bits only) to the sequencer
SP1_SQ_valid	SP1→SQ	1	Data valid
SP2_SQ_const_addr	SP2→SQ	36	Constant address load / predicate vector load (4 bits only) to the sequencer
SP2_SQ_valid	SP2→SQ	1	Data valid
SP3_SQ_const_addr	SP3→SQ	36	Constant address load / predicate vector load (4 bits only) to the sequencer
SP3_SQ_valid	SP3→SQ	1	Data valid

26.1.15 *SQ to SPx: constant broadcast*

Name	Direction	Bits	Description
SQ_SPx_constant	SQ→SPx	128	Constant broadcast

26.1.16 *SP0 to SQ: Kill vector load*

Name	Direction	Bits	Description
SP0_SQ_kill_vect	SP0→SQ	4	Kill vector load
SP1_SQ_kill_vect	SP1→SQ	4	Kill vector load
SP2_SQ_kill_vect	SP2→SQ	4	Kill vector load
SP3_SQ_kill_vect	SP3→SQ	4	Kill vector load

26.1.17 *SQ to CP: RBBM bus*

Name	Direction	Bits	Description
SQ_RBB_rs	SQ→CP	1	Read Strobe
SQ_RBB_rd	SQ→CP	32	Read Data
SQ_RBBM_nrtrtr	SQ→CP	1	Optional
SQ_RBBM_rtr	SQ→CP	1	Real-Time (Optional)

26.1.18 *CP to SQ: RBBM bus*

Name	Direction	Bits	Description
rbbm_we	CP→SQ	1	Write Enable
rbbm_a	CP→SQ	15	Address -- Upper Extent is TBD (16:2)
rbbm_wd	CP→SQ	32	Data
rbbm_be	CP→SQ	4	Byte Enables
rbbm_re	CP→SQ	1	Read Enable
rbb_rs0	CP→SQ	1	Read Return Strobe 0
rbb_rs1	CP→SQ	1	Read Return Strobe 1
rbb_rd0	CP→SQ	32	Read Data 0
rbb_rd1	CP→SQ	32	Read Data 0
RBBM_SQ_soft_reset	CP→SQ	1	Soft Reset

27. Examples of program executions

27.1.1 *Sequencer Control of a Vector of Vertices*

1. PA sends a vector of 64 vertices (actually vertex indices – 32 bits/index for 2048 bit total) to the RE's Vertex FIFO
   - state pointer as well as tag into position cache is sent along with vertices
   - space was allocated in the position cache for transformed position before the vector was sent
   - **also before the vector is sent to the RE, the CP has loaded the global instruction store with the vertex shader program (using the MH?)**
   - The vertex program is assumed to be loaded when we receive the vertex vector.
     - the SEQ then accesses the IS base for this shader using the local state pointer (provided to all sequencers by the RBBM when the CP is done loading the program)
2. SEQ arbitrates between the Pixel FIFO and the Vertex FIFO – basically the Vertex FIFO always has priority
   - at this point the vector is removed from the Vertex FIFO
   - the arbiter is not going to select a vector to be transformed if the parameter cache is full unless the pipe as nothing else to do (ie no pixels are in the pixel fifo).
3. SEQ allocates space in the SP register file for index data plus GPRs used by the program
   - the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer that came down with the vertices
   - SEQ will not send vertex data until space in the register file has been allocated
4. SEQ sends the vector to the SP register file over the RE_SP interface (which has a bandwidth of 2048 bits/cycle)
   - the 64 vertex indices are sent to the 64 register files over 4 cycles
     - RF0 of SU0, SU1, SU2, and SU3 is written the first cycle
     - RF1 of SU0, SU1, SU2, and SU3 is written the second cycle
     - RF2 of SU0, SU1, SU2, and SU3 is written the third cycle
     - RF3 of SU0, SU1, SU2, and SU3 is written the fourth cycle
   - the index is written to the least significant 32 bits **(floating point format?) (what about compound indices)** of the 128-bit location within the register file (w); the remaining data bits are set to zero (x, y, z)
5. SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of fetch state machine 0, or TSM0 FIFO)
   - the control packet contains the state pointer, the tag to the position cache and a register file base pointer.
6. TSM0 accepts the control packet and fetches the instructions for fetch clause 0 from the global instruction store
   - TSM0 was first selected by the TSM arbiter before it could start
7. all instructions of fetch clause 0 are issued by TSM0
8. the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)
   - TSM0 does not wait for requests made to the Fetch Unit to complete; it passes the register file write index for the fetch data to the TU, which will write the data to the RF as it is received
   - once the TU has written all the data to the register files, it increments a counter that is associated with ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go ahead start to execute the ALU clause
9. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store
10. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of fetch state machine 1, or TSM1 FIFO)
11. the control packet continues to travel down the path of reservation stations until all clauses have been executed
    - position can be exported in ALU clause 3 (or 4?); the data (and the tag) is sent over a position bus (which is shared with all four shader pipes) back to the PA's position cache
    - A parameter cache pointer is also sent along with the position data. This tells to the PA where the data is going to be in the parameter cache.
      - there is a position export FIFO in the SP that buffers position data before it gets sent back to the PA

    - the ASM arbiter will prevent a packet from starting an exporting clause if the position export FIFO is full
- parameter data is exported in clause 7 (as well as position data if it was not exported earlier)
    - parameter data is sent to the Parameter Cache over a dedicated bus
    - the SEQ allocates storage in the Parameter Cache, and the SEQ deallocates that space when there is no longer a need for the parameters (it is told by the PA when using a token).
    - the ASM arbiter will prevent a packet from starting on ASM7 if the parameter cache (or the position buffer if position is being exported) is full

12. after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program

27.1.2 *Sequencer Control of a Vector of Pixels*

1. **As with vertex shader programs, pixel shaders are loaded into the global instruction store by the CP**
    - At this point it is assumed that the pixel program is loaded into the instruction store and thus ready to be read.
2. the RE's Pixel FIFO is loaded with the barycentric coordinates for pixel quads by the detailed walker
    - the state pointer and the LOD correction bits are also placed in the Pixel FIF0
    - the Pixel FIFO is wide enough to source four quad's worth of barycentrics per cycle
3. SEQ arbitrates between Pixel FIFO and Vertex FIFO – when there are no vertices pending OR there is no space left in the register files for vertices, the Pixel FIFO is selected
4. SEQ allocates space in the SP register file for all the GPRs used by the program
    - the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer
    - SEQ will not allow interpolated data to be sent to the shader until space in the register file has been allocated
5. SEQ controls the transfer of interpolated data to the SP register file over the RE_SP interface (which has a bandwidth of 2048 bits/cycle). See interpolated data bus diagrams for details.
6. SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of fetch state machine 0, or TSM0 FIFO)
    - note that there is a separate set of reservation stations/arbiters/state machines for vertices and for pixels
    - the control packet contains the state pointer, the register file base pointer, and the LOD correction bits
    - all other information (such as quad address for example) travels in a separate FIFO
7. TSM0 accepts the control packet and fetches the instructions for fetch clause 0 from the global instruction store
    - TSM0 was first selected by the TSM arbiter before it could start
8. all instructions of fetch clause 0 are issued by TSM0
9. the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)
    - TSM0 does not wait for fetch requests made to the Fetch Unit to complete; it passes the register file write index for the fetch data to the TU, which will write the data to the RF as it is received
    - once the TU has written all the data for a particular clause to the register files, it increments a counter that is associated with the ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go ahead and pop the FIFO and start to execute the ALU clause
10. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store
11. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of fetch state machine 1, or TSM1 FIFO)
12. the control packet continues to travel down the path of reservation stations until all clauses have been executed
    - pixel data is exported in the last ALU clause (clause 7)
        - it is sent to an output FIFO where it will be picked up by the render backend
        - the ASM arbiter will prevent a packet from starting on ASM7 if the output FIFO is full
13. after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program

27.1.3 *Notes*

14. The state machines and arbiters will operate ahead of time so that they will be able to immediately start the real threads or stall.
15. The register file base pointer for a vector needs to travel with the vector through the reservation stations, but the instruction store base pointer does not – this is because the RF pointer is different for all threads, but the IS pointer is only different for each state and thus can be accessed via the state pointer.

28. Open issues

Need to do some testing on the size of the register file as well as on the register file allocation method (dynamic VS static).

Saving power?

Parameter caches in SX?

Using both IJ buffers for center + centroid interpolation?

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 2015~~25~~ March, 20024 March	GEN-CXXXXX-REVA	1 of 52

Author: Laurent Lefebvre

Issue To:	**Copy No:**

R400 Sequencer Specification

SQ

Version 1.10~~8~~

Overview: This is an architectural specification for the R400 Sequencer block (SEQ). It provides an overview of the required capabilities and expected uses of the block. It also describes the block interfaces, internal sub-blocks, and provides internal state diagrams.

AUTOMATICALLY UPDATED FIELDS:
Document Location: C:\perforce\r400\doc_lib\design\blocks\sq\R400_Sequencer.doc
Current Intranet Search Title: R400 Sequencer Specification

APPROVALS	
Name/Dept	Signature/Date

Remarks:

THIS DOCUMENT CONTAINS CONFIDENTIAL INFORMATION THAT COULD BE SUBSTANTIALLY DETRIMENTAL TO THE INTEREST OF ATI TECHNOLOGIES INC. THROUGH UNAUTHORIZED USE OR DISCLOSURE.

Table Of Contents

1. OVERVIEW .... **8~~6~~**
1.1 Top Level Block Diagram .... 10~~8~~
1.2 Data Flow graph (SP) .... 12~~10~~
1.3 Control Graph .... 13~~11~~
2. INTERPOLATED DATA BUS .... **13~~11~~**
3. INSTRUCTION STORE .... **16~~14~~**
4. SEQUENCER INSTRUCTIONS .... **18~~16~~**
5. CONSTANT STORES .... **18~~16~~**
5.1 Memory organizations .... 18~~16~~
5.2 Management of the Control Flow Constants .... 18~~16~~
5.3 Management of the re-mapping tables .... 18~~16~~
5.3.1 R400 Constant management .... 18~~16~~
5.3.2 Proposal for R400LE constant management .... 19~~17~~
5.3.3 Dirty bits .... 21~~19~~
5.3.4 Free List Block .... 21~~19~~
5.3.5 De-allocate Block .... 22~~20~~
5.3.6 Operation of Incremental model .... 22~~20~~
5.4 Constant Store Indexing .... 22~~20~~
5.5 Real Time Commands .... 23~~21~~
5.6 Constant Waterfalling .... 23~~21~~
6. LOOPING AND BRANCHES .... **24~~22~~**
6.1 The controlling state. .... 24~~22~~
6.2 The Control Flow Program .... 24~~22~~
6.3 Data dependant predicate instructions .... 26~~24~~
6.4 HW Detection of PV,PS .... 27~~25~~
6.5 Register file indexing .... 27~~25~~
6.6 Predicated Instruction support for Texture clauses .... 27~~25~~
6.7 Debugging the Shaders .... 28~~25~~
6.7.1 Method 1: Debugging registers .... 28~~25~~
6.7.2 Method 2: Exporting the values in the GPRs (12) .... 28~~26~~
7. PIXEL KILL MASK .... **28~~26~~**
8. MULTIPASS VERTEX SHADERS (HOS) .... **29~~26~~**
9. REGISTER FILE ALLOCATION .... **29~~26~~**
10. FETCH ARBITRATION .... **30~~28~~**
11. ALU ARBITRATION .... **30~~28~~**
12. HANDLING STALLS .... **31~~29~~**
13. CONTENT OF THE RESERVATION STATION FIFOS .... **31~~29~~**
14. THE OUTPUT FILE .... **31~~29~~**
15. IJ FORMAT .... **31~~29~~**
15.1 Interpolation of constant attributes .... 32~~30~~
16. STAGING REGISTERS .... **32~~30~~**
17. THE PARAMETER CACHE .... **34~~32~~**
18. VERTEX POSITION EXPORTING .... **35~~32~~**

19. EXPORTING ARBITRATION .... **35~~32~~**
20. EXPORT TYPES .... **35~~32~~**
20.1 Vertex Shading .... 35~~32~~
20.2 Pixel Shading .... 35~~33~~
21. SPECIAL INTERPOLATION MODES .... **36~~33~~**
21.1 Real time commands .... 36~~33~~
21.2 Sprites/ XY screen coordinates/ FB information .... 36~~33~~
21.3 Auto generated counters .... 36~~34~~
21.3.1 Vertex shaders .... 37~~34~~
21.3.2 Pixel shaders .... 37~~34~~
22. STATE MANAGEMENT .... **37~~34~~**
22.1 Parameter cache synchronization .... 37~~34~~
23. XY ADDRESS IMPORTS .... **38~~35~~**
23.1 Vertex indexes imports .... 38~~35~~
24. REGISTERS .... **38~~35~~**
24.1 Control .... 38~~35~~
24.2 Context .... 38~~35~~
25. DEBUG REGISTERS .... **39~~36~~**
25.1 Context .... 39~~36~~
25.2 Control .... 39~~36~~
26. INTERFACES .... **39~~36~~**
26.1 External Interfaces .... 39~~36~~
26.1.1 SC to SQ : IJ Control bus .... 39~~37~~
26.1.2 SQ to SP: Interpolator bus .... 42~~37~~
26.1.3 SQ to SX: Interpolator bus .... 42~~37~~
26.1.4 SQ to SP: Staging Register Data .... 42~~38~~
26.1.5 PA to SQ : Vertex interface .... 43~~38~~
26.1.6 SQ to CP: State report .... 46~~41~~
26.1.7 SQ to SX: Control bus .... 46~~41~~
26.1.8 SX to SQ : Output file control .... 46~~41~~
26.1.9 SQ to TP: Control bus .... 46~~41~~
26.1.10 TP to SQ: Texture stall .... 47~~42~~
26.1.11 SQ to SP: Texture stall .... 47~~42~~
26.1.12 SQ to SP: GPR and auto counter .... 47~~42~~
26.1.13 SQ to SPx: Instructions .... 48~~43~~
26.1.14 SP to SQ: Constant address load/ Predicate Set .... 48~~44~~
26.1.15 SQ to SPx: constant broadcast .... 49~~44~~
26.1.16 SP0 to SQ: Kill vector load .... 49~~44~~
26.1.17 SQ to CP: RBBM bus .... 49~~44~~
26.1.18 CP to SQ: RBBM bus .... 49~~44~~
27. EXAMPLES OF PROGRAM EXECUTIONS .... **49~~45~~**
27.1.1 Sequencer Control of a Vector of Vertices .... 49~~45~~

27.1.2 Sequencer Control of a Vector of Pixels .... 50~~46~~

27.1.3 Notes .... 51~~47~~

28. OPEN ISSUES .... 51~~47~~

~~**1. OVERVIEW .... 6**~~

~~1.1 Top Level Block Diagram .... 8~~

~~1.2 Data Flow graph (SP) .... 10~~

~~1.3 Control Graph .... 11~~

~~**2. INTERPOLATED DATA BUS .... 11**~~

~~**3. INSTRUCTION STORE .... 14**~~

~~**4. SEQUENCER INSTRUCTIONS .... 16**~~

~~**5. CONSTANT STORES .... 16**~~

~~5.1 Memory organizations .... 16~~

~~5.2 Management of the re-mapping tables .... 16~~

~~5.2.1 Dirty bits .... 18~~

~~5.2.2 Free List Block .... 18~~

~~5.2.3 De-allocate Block .... 19~~

~~5.2.4 Operation of Incremental model .... 19~~

~~5.3 Constant Store Indexing .... 19~~

~~5.4 Real Time Commands .... 20~~

~~5.5 Constant Waterfalling .... 20~~

~~**6. LOOPING AND BRANCHES .... 21**~~

~~6.1 The controlling state. .... 21~~

~~6.2 The Control Flow Program .... 21~~

~~6.3 Data dependant predicate instructions .... 23~~

~~6.4 HW Detection of PV,PS .... 24~~

~~6.5 Register file indexing .... 24~~

~~6.6 Predicated Instruction support for Texture clauses .... 24~~

~~6.7 Debugging the Shaders .... 25~~

~~6.7.1 Method 1: Debugging registers .... 25~~

~~6.7.2 Method 2: Exporting the values in the GPRs (12) .... 25~~

~~**7. PIXEL KILL MASK .... 25**~~

~~**8. MULTIPASS VERTEX SHADERS (HOS) .... 26**~~

~~**9. REGISTER FILE ALLOCATION .... 26**~~

~~**10. FETCH ARBITRATION .... 27**~~

~~**11. ALU ARBITRATION .... 27**~~

~~**12. HANDLING STALLS .... 28**~~

~~**13. CONTENT OF THE RESERVATION STATION FIFOS .... 28**~~

~~**14. THE OUTPUT FILE .... 28**~~

~~**15. IJ FORMAT .... 28**~~

~~15.1 Interpolation of constant attributes .... 29~~

~~**16. STAGING REGISTERS .... 29**~~

~~**17. THE PARAMETER CACHE .... 31**~~

~~**18. VERTEX POSITION EXPORTING .... 31**~~

~~**19. EXPORTING ARBITRATION .... 31**~~

~~**20. EXPORT TYPES .... 31**~~

~~20.1 Vertex Shading .... 31~~

~~20.2 Pixel Shading .... 32~~
~~21. SPECIAL INTERPOLATION MODES .... 32~~
~~21.1 Real time commands .... 32~~
~~21.2 Sprites/ XY screen coordinates/ FB information .... 32~~
~~21.3 Auto generated counters .... 33~~
~~21.3.1 Vertex shaders .... 33~~
~~21.3.2 Pixel shaders .... 33~~
~~22. STATE MANAGEMENT .... 33~~
~~22.1 Parameter cache synchronization .... 33~~
~~23. XY ADDRESS IMPORTS .... 34~~
~~23.1 Vertex indexes imports .... 34~~
~~24. REGISTERS .... 34~~
~~24.1 Control .... 34~~
~~24.2 Context .... 34~~
~~25. DEBUG REGISTERS .... 35~~
~~25.1 Context .... 35~~
~~26. INTERFACES .... 35~~
~~26.1 External Interfaces .... 35~~
~~26.1.1 SC to SQ : IJ Control bus .... 36~~
~~26.1.2 SQ to SP: Interpolator bus .... 36~~
~~26.1.3 SQ to SP: Parameter Cache Read control bus .... 36~~
~~26.1.4 SQ to SX: Parameter Cache Mux control Bus .... 37~~
~~26.1.5 SQ to SP: Staging Register Data .... 37~~
~~26.1.6 PA to SQ : Vertex interface .... 37~~
~~26.1.7 SQ to CP: State report .... 41~~
~~26.1.8 SQ to SX: Control bus .... 41~~
~~26.1.9 SX to SQ : Output file control .... 41~~
~~26.1.10 SQ to TP: Control bus .... 41~~
~~26.1.11 TP to SQ: Texture stall .... 42~~
~~26.1.12 SQ to SP: Texture stall .... 42~~
~~26.1.13 SQ to SP: GPR, Parameter cache control and auto counter .... 42~~
~~26.1.14 SQ to SPx: Instructions .... 43~~
~~26.1.15 SP to SQ: Constant address load .... 44~~
~~26.1.16 SQ to SPx: constant broadcast .... 44~~
~~26.1.17 SP0 to SQ: Kill vector load .... 44~~
~~26.1.18 SQ to CP: RBBM bus .... 44~~
~~26.1.19 CP to SQ: RBBM bus .... 44~~
~~27. EXAMPLES OF PROGRAM EXECUTIONS .... 44~~
~~27.1.1 Sequencer Control of a Vector of Vertices .... 44~~
~~27.1.2 Sequencer Control of a Vector of Pixels .... 46~~
~~27.1.3 Notes .... 46~~

~~28. OPEN ISSUES..47~~

Revision Changes:

Rev 0.1 (Laurent Lefebvre) Date: May 7, 2001	First draft.
Rev 0.2 (Laurent Lefebvre) Date : July 9, 2001	Changed the interfaces to reflect the changes in the SP. Added some details in the arbitration section.
Rev 0.3 (Laurent Lefebvre) Date : August 6, 2001	Reviewed the Sequencer spec after the meeting on August 3, 2001.
Rev 0.4 (Laurent Lefebvre) Date : August 24, 2001	Added the dynamic allocation method for register file and an example (written in part by Vic) of the flow of pixels/vertices in the sequencer.
Rev 0.5 (Laurent Lefebvre) Date : September 7, 2001	Added timing diagrams (Vic)
Rev 0.6 (Laurent Lefebvre) Date : September 24, 2001	Changed the spec to reflect the new R400 architecture. Added interfaces.
Rev 0.7 (Laurent Lefebvre) Date : October 5, 2001	Added constant store management, instruction store management, control flow management and data dependant predication.
Rev 0.8 (Laurent Lefebvre) Date : October 8, 2001	Changed the control flow method to be more flexible. Also updated the external interfaces.
Rev 0.9 (Laurent Lefebvre) Date : October 17, 2001	Incorporated changes made in the 10/18/01 control flow meeting. Added a NOP instruction, removed the conditional_execute_or_jump. Added debug registers.
Rev 1.0 (Laurent Lefebvre) Date : October 19, 2001	Refined interfaces to RB. Added state registers.
Rev 1.1 (Laurent Lefebvre) Date : October 26, 2001	Added SEQ→SP0 interfaces. Changed delta precision. Changed VGT→SP0 interface. Debug Methods added.
Rev 1.2 (Laurent Lefebvre) Date : November 16, 2001	Interfaces greatly refined. Cleaned up the spec.
Rev 1.3 (Laurent Lefebvre) Date : November 26, 2001	Added the different interpolation modes.
Rev 1.4 (Laurent Lefebvre) Date : December 6, 2001	Added the auto incrementing counters. Changed the VGT→SQ interface. Added content on constant management. Updated GPRs.
Rev 1.5 (Laurent Lefebvre) Date : December 11, 2001	Removed from the spec all interfaces that weren't directly tied to the SQ. Added explanations on constant management. Added PA→SQ synchronization fields and explanation.
Rev 1.6 (Laurent Lefebvre) Date : January 7, 2002	Added more details on the staging register. Added detail about the parameter caches. Changed the call instruction to a Conditionnal_call instruction. Added details on constant management and updated the diagram.
Rev 1.7 (Laurent Lefebvre) Date : February 4, 2002	Added Real Time parameter control in the SX interface. Updated the control flow section.
Rev 1.8 (Laurent Lefebvre) Date : March 4, 2002	New interfaces to the SX block. Added the end of clause modifier, removed the end of clause instructions.
Rev 1.9 (Laurent Lefebvre) Date : March 18, 2002	Rearangement of the CF instruction bits in order to ensure byte alignment.
Rev 1.10 (Laurent Lefebvre) Date : March 25, 2002	Updated the interfaces and added a section on exporting rules.

1. Overview

The sequencer is based on the R300 design. It chooses two ALU clauses and a fetch clause to execute, and executes all of the instructions in a clause before looking for a new clause of the same type. Two ALU clauses are executed interleaved to hide the ALU latency. Each vector will have eight fetch and eight ALU clauses, but clauses do not need to contain instructions. A vector of pixels or vertices ping-pongs along the sequencer FIFO, bouncing from fetch reservation station to alu reservation station. A FIFO exists between each reservation stage, holding up vectors until the vector currently occupying a reservation station has left. A vector at a reservation station can be chosen to execute. The sequencer looks at all eight alu reservation stations to choose an alu clause to execute and all eight fetch stations to choose a fetch clause to execute. The arbitrator will give priority to clauses/reservation stations closer to the bottom of the pipeline. It will not execute an alu clause until the fetch fetches initiated by the previous fetch clause have completed. There are two separate sets of reservation stations, one for pixel vectors and one for vertices vectors. This way a pixel can pass a vertex and a vertex can pass a pixel.

To support the shader pipe the sequencer also contains the shader instruction cache, constant store, control flow constants and texture state. The four shader pipes also execute the same instruction thus there is only one sequencer for the whole chip.

The sequencer first arbitrates between vectors of 64 vertices that arrive directly from primitive assembly and vectors of 16 quads (64 pixels) that are generated in the scan converter.

The vertex or pixel program specifies how many GPRs it needs to execute. The sequencer will not start the next vector until the needed space is available in the GPRs.

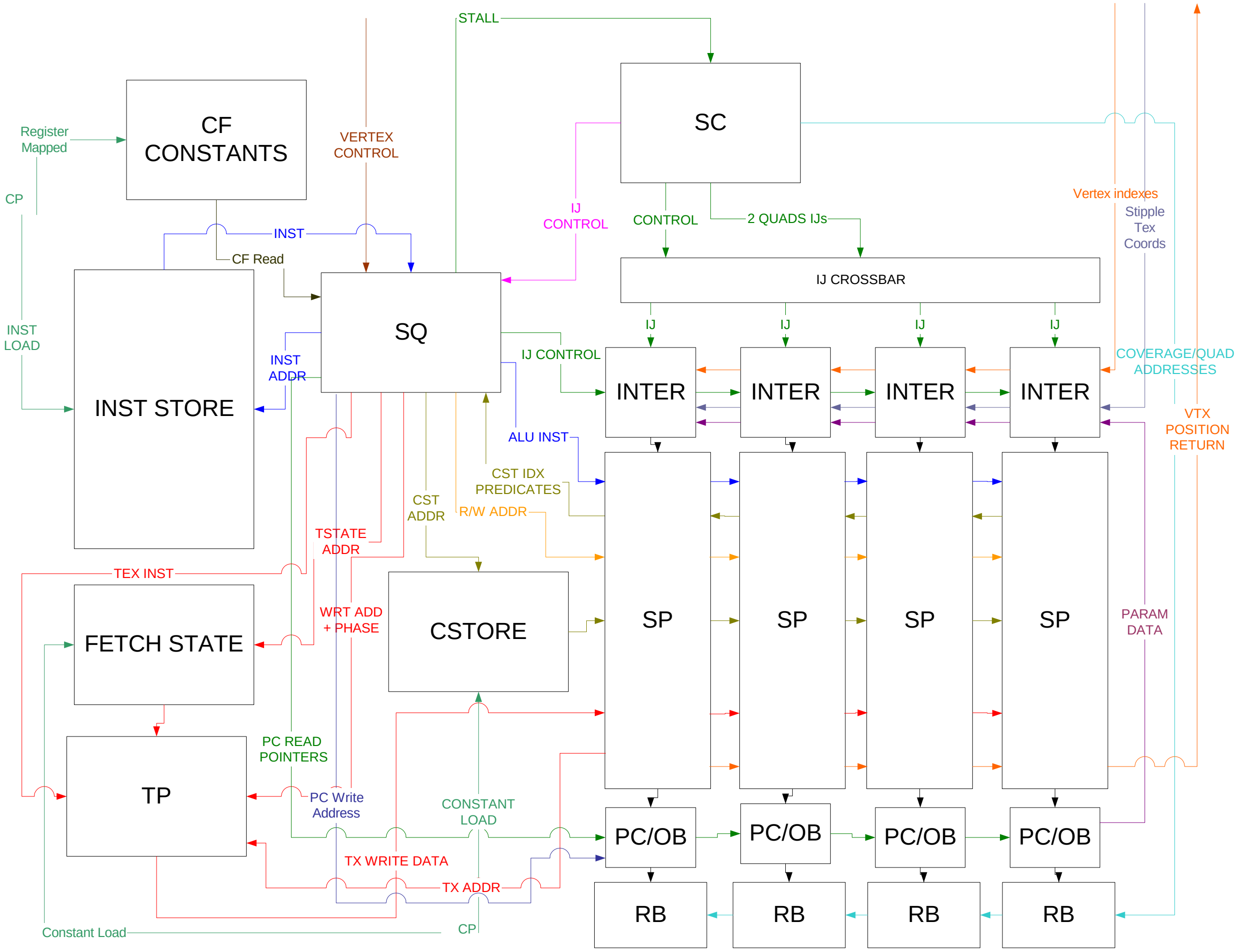
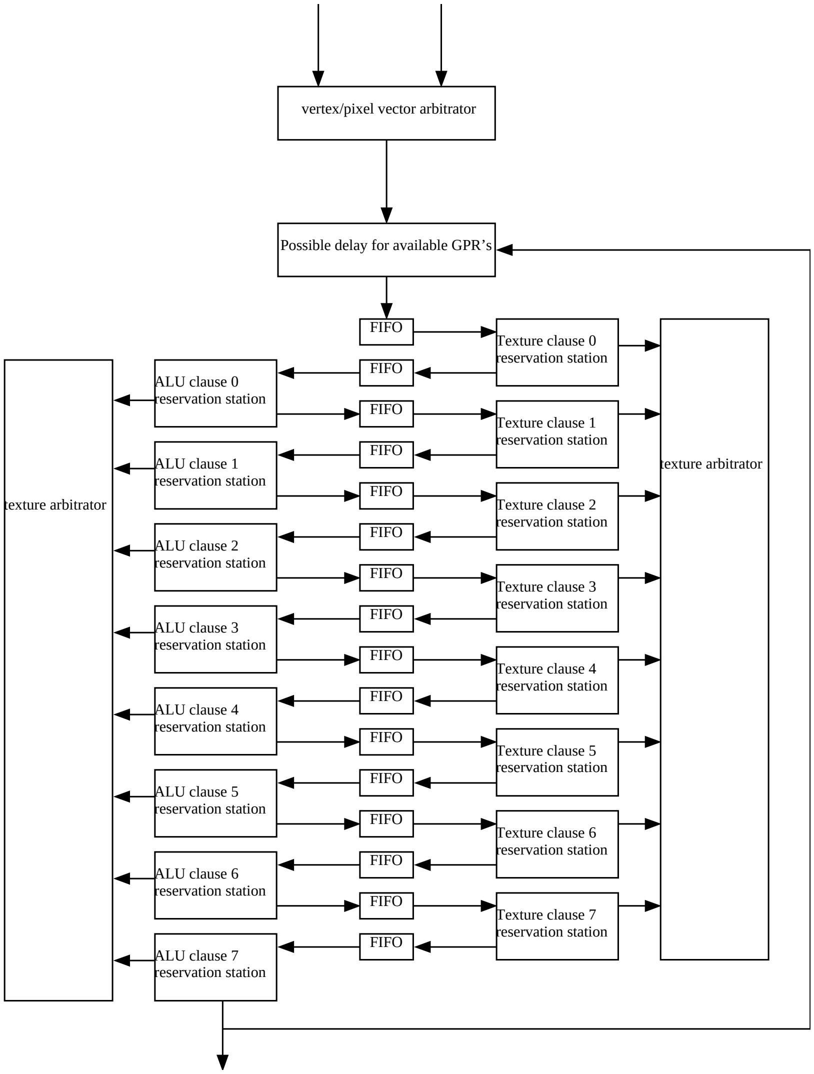


Figure 1: General Sequencer overview

1.1 Top Level Block Diagram



vertex/pixel vector arbitrator
Possible delay for available GPR's
FIFO
Texture clause 0 reservation station
Texture clause 1 reservation station
Texture clause 2 reservation station
Texture clause 3 reservation station
Texture clause 4 reservation station
Texture clause 5 reservation station
Texture clause 6 reservation station
Texture clause 7 reservation station
texture arbitrator
ALU clause 0 reservation station
ALU clause 1 reservation station
ALU clause 2 reservation station
ALU clause 3 reservation station
ALU clause 4 reservation station
ALU clause 5 reservation station
ALU clause 6 reservation station
ALU clause 7 reservation station
texture arbitrator

Figure 2: Reservation stations and arbiters

There are two sets of the above figure, one for vertices and one for pixels.

Depending on the arbitration state, the sequencer will either choose a vertex or a pixel packet. The control packet consists of 3 bits of state, 7 bits for the base address of the Shader program and some information on the coverage to determine fetch LOD plus other various small state bits.

On receipt of a packet, the input state machine (not pictured but just before the first FIFO) allocated enough space in the GPRs to store the interpolated values and temporaries. Following this, the barycentric coordinates (and XY screen position if needed) are sent to the interpolator, which will use them to interpolate the parameters and place the results into the GPRs. Then, the input state machine stacks the packet in the first FIFO.

On receipt of a command, the level 0 fetch machine issues a fetch request to the TP and corresponding GPR address for the fetch address (ta). A small command (tcmd) is passed to the fetch system identifying the current level number (0) as well as the GPR write address for the fetch return data. One fetch request is sent every 4 clocks causing the texturing of sixteen 2x2s worth of data (or 64 vertices). Once all the requests are sent the packet is put in FIFO 1.

Upon receipt of the return data, the fetch unit writes the data to the register file using the write address that was provided by the level 0 fetch machine and sends the clause number (0) to the level 0 fetch state machine to signify that the write is done and thus the data is ready. Then, the level 0 fetch machine increments the counter of FIFO 1 to signify to the ALU 0 that the data is ready to be processed.

On receipt of a command, the level 0 ALU machine first decrements the input FIFO 1 counter and then issues a complete set of level 0 shader instructions. For each instruction, the ALU state machine generates 3 source addresses, one destination address and an instruction. Once the last instruction has been issued, the packet is put into FIFO 2.

There will always be two active ALU clauses at any given time (and two arbiters). One arbiter will arbitrate over the odd instructions (4 clocks cycles) and the other one will arbitrate over the even instructions (4 clocks cycles). The only constraints between the two arbiters is that they are not allowed to pick the same clause number as the other one is currently working on if the packet is not of the same type (render state).

If the packet is a vertex packet, upon reaching ALU clause 3, it can export the position if the position is ready. So the arbiter must prevent ALU clause 3 to be selected if the positional buffer is full (or can't be accessed). Along with the positional data, if needed the sprite size and/or edge flags can also be sent.

A special case is for multipass vertex shaders, which can export 12 parameters per last 6 clauses to the output buffer. If the output buffer is full or doesn't have enough space the sequencer will prevent such a vertex group to enter an exporting clause.

Multipass pixel shaders can export 12 parameters to memory from the last clause only (7).

All other clauses process in the same way until the packet finally reaches the last ALU machine (7).

Only one pair of interleaved ALU state machines may have access to the register file address bus or the instruction decode bus at one time. Similarly, only one fetch state machine may have access to the register file address bus at one time. Arbitration is performed by three arbiter blocks (two for the ALU state machines and one for the fetch state machines). The arbiters always favor the higher number state machines, preventing a bunch of half finished jobs from clogging up the register files.

1.2 Data Flow graph (SP)

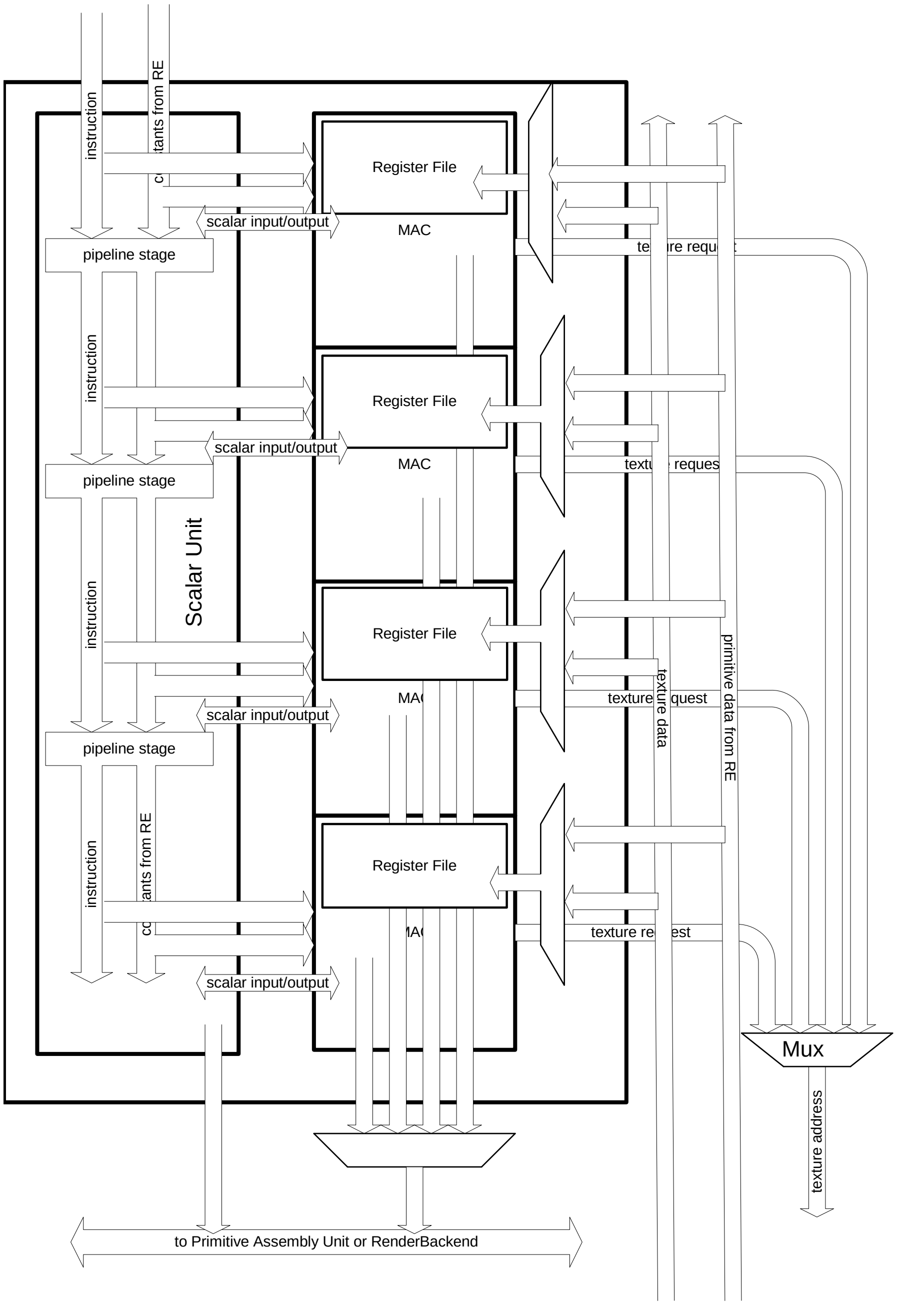


Figure 3: The shader Pipe

The gray area represents blocks that are replicated 4 times per shader pipe (16 times on the overall chip).

1.3 Control Graph

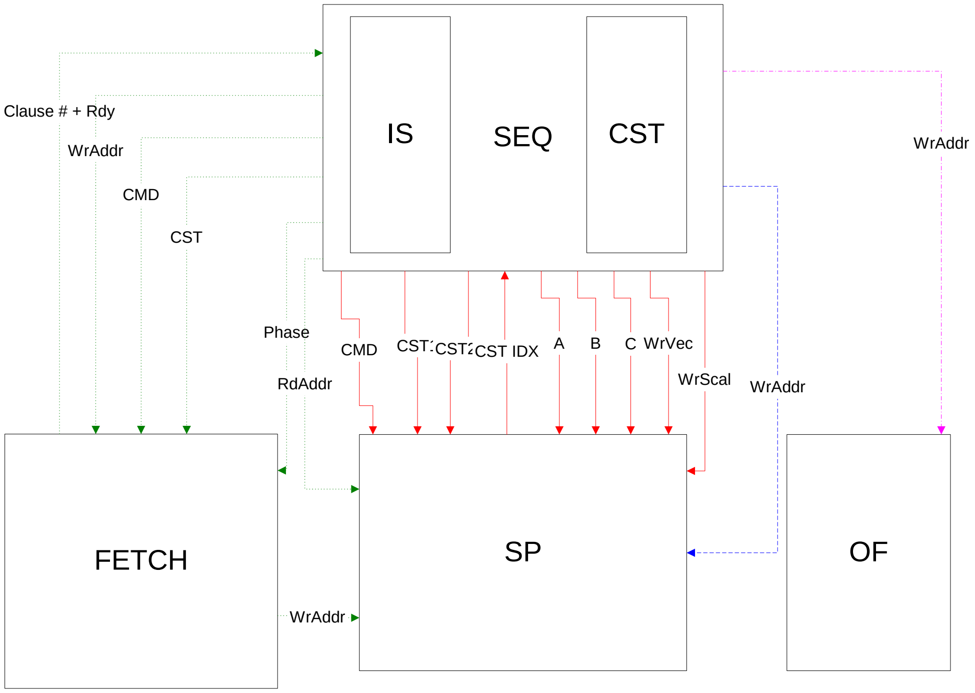


Figure 4: Sequencer Control interfaces

In green is represented the Fetch control interface, in red the ALU control interface, in blue the Interpolated/Vector control interface and in purple is the output file control interface.

2. Interpolated data bus

The interpolators contain an IJ buffer to pack the information as much as possible before writing it to the register file.

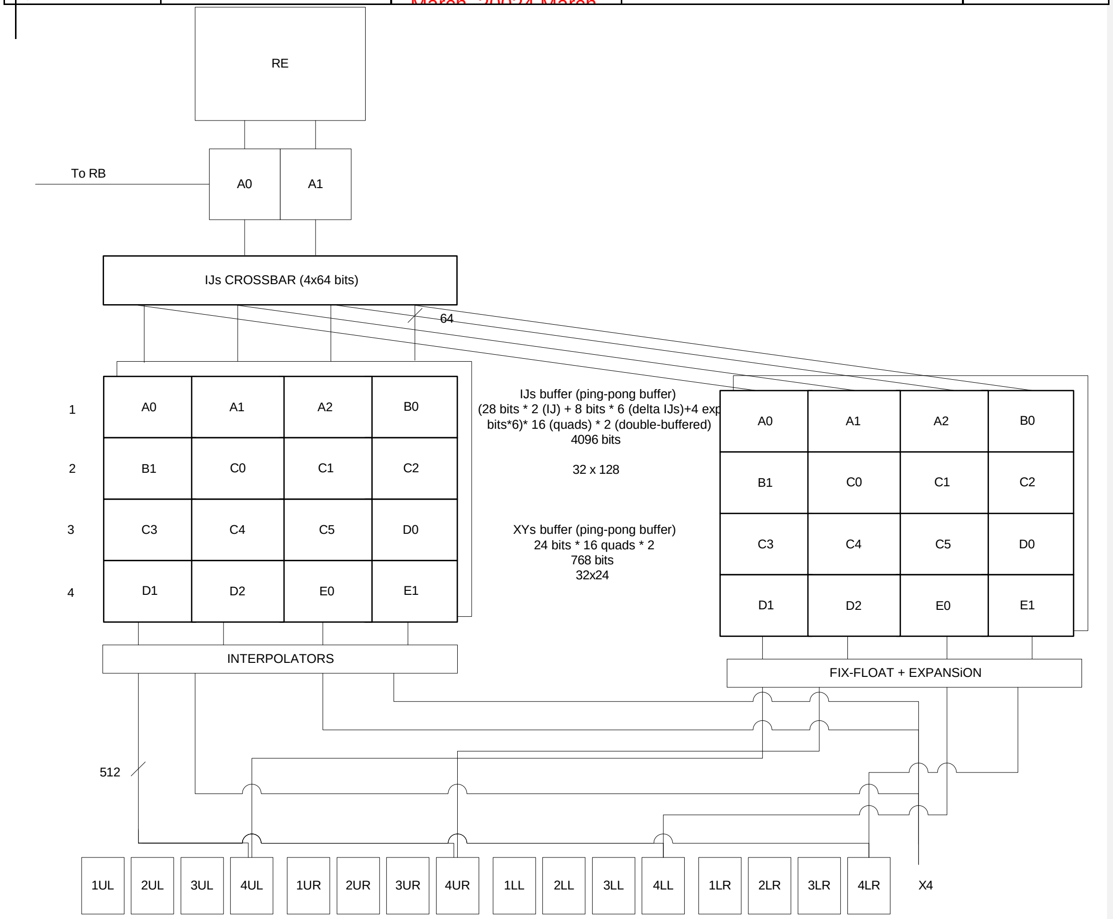
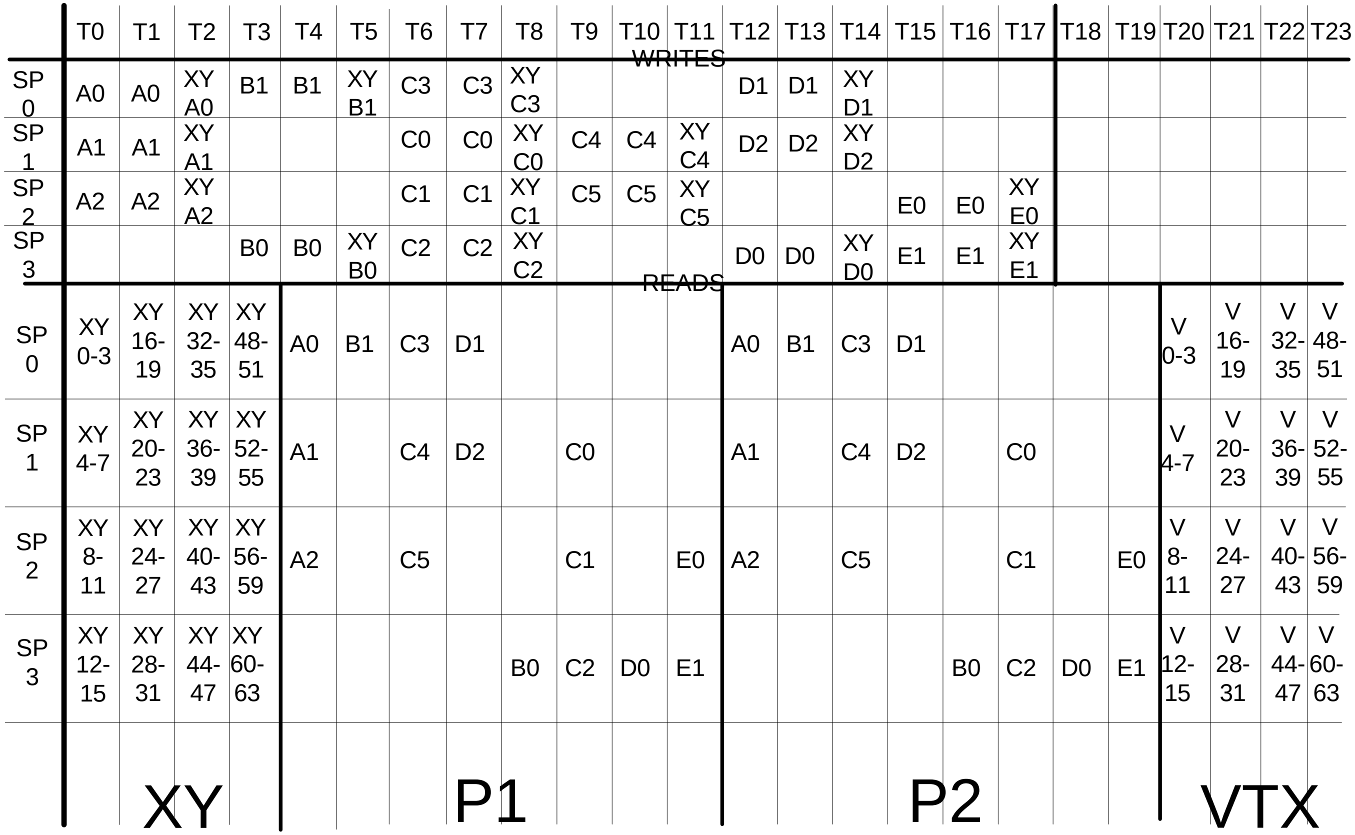


Figure 5: Interpolation buffers



	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21	T22	T23
SP 0	A0	A0	XY A0	B1	B1	XY B1	C3	C3	XY C3				D1	D1	XY D1									
SP 1	A1	A1	XY A1				C0	C0	XY C0	C4	C4	XY C4	D2	D2	XY D2									
SP 2	A2	A2	XY A2				C1	C1	XY C1	C5	C5	XY C5				E0	E0	XY E0						
SP 3				B0	B0	XY B0	C2	C2	XY C2				D0	D0	XY D0	E1	E1	XY E1						
SP 0	XY 0-3	XY 16-19	XY 32-35	XY 48-51	A0	B1	C3	D1					A0	B1	C3	D1					V 0-3	V 16-19	V 32-35	V 48-51
SP 1	XY 4-7	XY 20-23	XY 36-39	XY 52-55	A1		C4	D2		C0			A1		C4	D2		C0			V 4-7	V 20-23	V 36-39	V 52-55
SP 2	XY 8-11	XY 24-27	XY 40-43	XY 56-59	A2		C5			C1		E0	A2		C5			C1		E0	V 8-11	V 24-27	V 40-43	V 56-59
SP 3	XY 12-15	XY 28-31	XY 44-47	XY 60-63					B0	C2	D0	E1					B0	C2	D0	E1	V 12-15	V 28-31	V 44-47	V 60-63

Figure 6: Interpolation timing diagram

Above is an example of a tile the sequencer might receive from the SC. The write side is how the data get stacked into the XY and IJ buffers, the read side is how the data is passed to the GPRs. The IJ information is packed in the IJ buffer 4 quads at a time or two clocks. The sequencer allows at any given time as many as four quads to interpolate a parameter. They all have to come from the same primitive. Then the sequencer controls the write mask to the GPRs to write the valid data in.

~~{ISSUE : Do we do the center + centroid approach using both IJ buffers?}~~

3. Instruction Store

There is going to be only one instruction store for the whole chip. It will contain 4096 instructions of 96 bits each.

It is likely to be a 1 port memory; we use 1 clock to load the ALU instruction, 1 clocks to load the Fetch instruction, 1 clock to load 2 control flow instructions and 1 clock to write instructions.

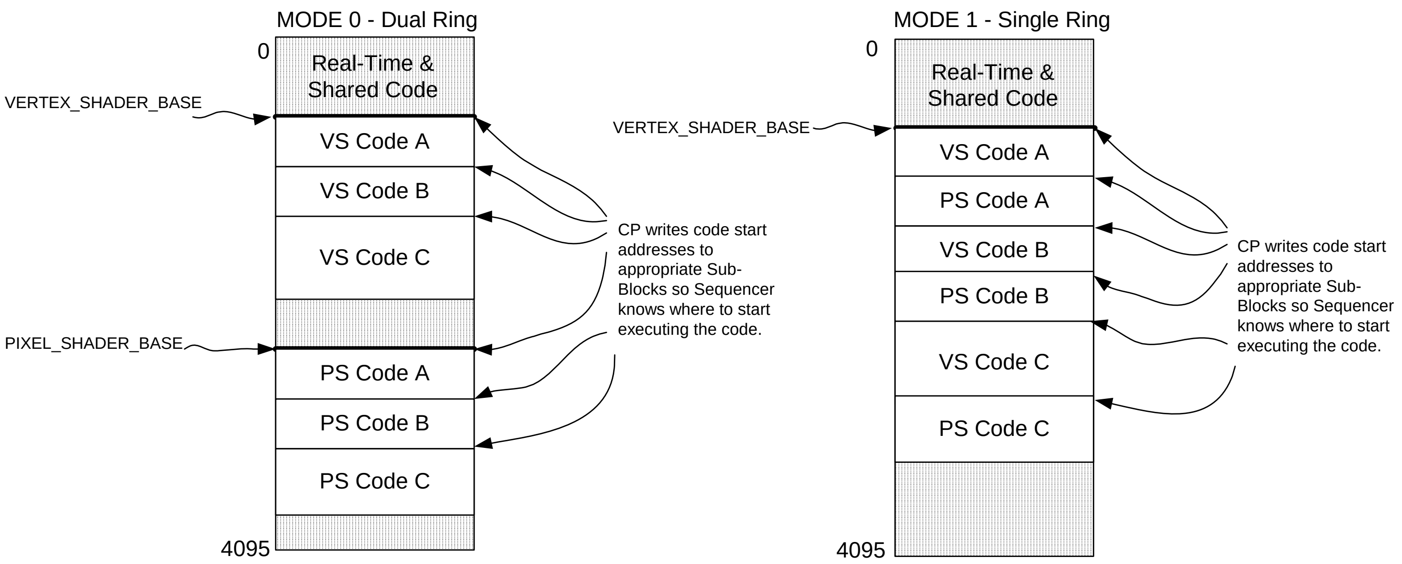
The instruction store is loaded by the CP thru the register mapped registers.

The next picture shows the various modes the CP can load the memory. The Sequencer has to keep track of the loading modes in order to wrap around the correct boundaries. The wrap-around points are arbitrary and they are specified in the VS_BASE and PIX_BASE control registers. The VS_BASE and PS_BASE context registers are used to specify for each context where its shader is in the instruction memory.

For the Real time commands the story is quite the same but for some small differences. There are no wrap-around points for real time so the driver must be careful not to overwrite regular shader data. The shared code (shared subroutines) uses the same path as real time.

R400 CP's Views of Instruction Memory

Updated: 11/14/2001
John A. Carey



CP writes code start addresses to appropriate Sub-Blocks so Sequencer knows where to start executing the code.

CP writes code start addresses to appropriate Sub-Blocks so Sequencer knows where to start executing the code.

Figure 7: The CP's view of the instruction memory

4. Sequencer Instructions

All control flow instructions and move instructions are handled by the sequencer only. The ALUs will perform NOPs during this time (MOV PV,PV, PS,PS) if they have nothing else to do.

5. Constant Stores

5.1 Memory organizations

A likely size for the ALU constant store is 1024x128 bits. The read BW from the ALU constant store is 128 bits/clock and the write bandwidth is 32 bits/clock (directed by the CP bus size not by memory ports).

The maximum logical size of the constant store for a given shader is 256 constants. Or 512 for the pixel/vertex shader pair. The size of the re-mapping table is 128 lines (each line addresses 4 constants). The write granularity is 4 constants or 512 bits. It takes 16 clocks to write the four constants. Real time requires 256 lines in the physical memory (this is physically register mapped).

The texture state is also kept in a similar memory. The size of this memory is ~~128x192~~ 320x96 bits (128 texture states for regular mode, 32 states for RT). The memory thus holds 128 texture states (192 bits per state). The logical size exposes 32 different states total, which are going to be shared between the pixel and the vertex shader. The size of the re-mapping table to for the texture state memory is 32 lines (each line addresses 1 texture state lines in the real memory). The CP write granularity is 1 texture state lines (or 192 bits). The driver sends 512 bits but the CP ignores the top 320 bits. It thus takes 6 clocks to write the texture state. Real time requires 32 lines in the physical memory (this is physically register mapped).

The control flow constant memory doesn't sit behind a renaming table. It is register mapped and thus the driver must reload its content each time there is a change in the control flow constants. Its size is 320*32 because it must hold 8 copies of the 32 dwords of control flow constants and the loop construct constants must be aligned.

The constant re-mapping tables for texture state and ALU constants are logically register mapped for regular mode and physically register mapped for RT operation.

5.2 Management of the Control Flow Constants

The control flow constants are register mapped, thus the CP writes to the according register to set the constant, the SQ decodes the address and writes to the block pointed by its current base pointer (CF_WR_BASE). On the read side, one level of indirection is used. A register (SQ_CONTEXT_MISC.CF_RD_BASE) keeps the current base pointer to the control flow block. This register is copied whenever there is a state change. Should the CP write to CF after the state change, the base register is updated with the (current pointer number +1)% number of states. This way, if the CP doesn't write to CF the state is going to use the previous CF constants.

5.3 Management of the re-mapping tables

5.3.1 *R400 Constant management*

The sequencer is responsible to manage two re-mapping tables (one for the constant store and one for the texture state). On a state change (by the driver), the sequencer will broadside copy the contents of its re-mapping tables to a new one. We have 8 different re-mapping tables we can use concurrently.

The constant memory update will be incremental, the driver only need to update the constants that actually changed between the two state changes.

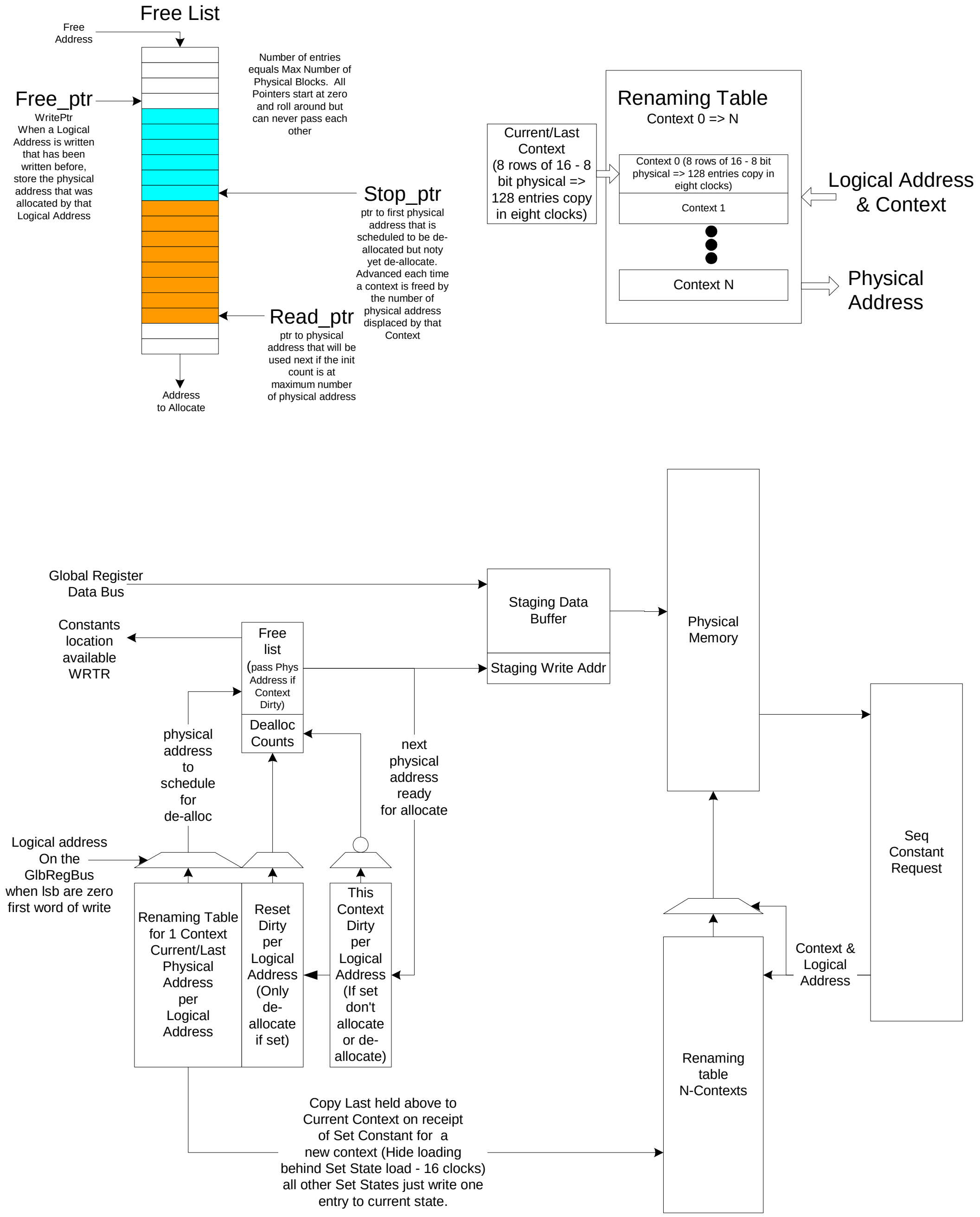
For this model to work in its simplest form, the requirement is that the physical memory MUST be at least twice as large as the logical address space + the space allocated for Real Time. In our case, since the logical address space

is 512 and the reserved RT space can be up to 256 entries, the memory must be of sizes 1280 and above. Similarly the size of the texture store must be of 32*2+32 = 96 entries and above.

5.3.2 *Proposal for R400LE constant management*

To make this scheme work with only 512+256 = 768 entries, upon reception of a CONTROL packet of state + 1, the sequencer would check for SQ_IDLE and PA_IDLE and if both are idle will erase the content of state to replace it with the new state (this is depicted in Figure 9: De-allocation mechanism~~Figure 9: De-allocation mechanism~~~~Figure 9: De-allocation mechanism~~). Note that in the case a state is cleared a value of 0 is written to the corresponding de-allocation counter location so that when the SQ is going to report a state change, nothing will be de-allocated upon the first report.

The second path sets all context dirty bits that were used in the current state to 1 (thus allowing the new state to reuse these physical addresses if needed).



Free List
Free Address
Free_ptr
WritePtr
When a Logical Address is written that has been written before, store the physical address that was allocated by that Logical Address
Number of entries equals Max Number of Physical Blocks. All Pointers start at zero and roll around but can never pass each other
Stop_ptr
ptr to first physical address that is scheduled to be de-allocated but noty yet de-allocate. Advanced each time a context is freed by the number of physical address displaced by that Context
Read_ptr
ptr to physical address that will be used next if the init count is at maximum number of physical address
Address to Allocate
Renaming Table
Context 0 => N
Current/Last Context (8 rows of 16 - 8 bit physical => 128 entries copy in eight clocks)
Context 0 (8 rows of 16 - 8 bit physical => 128 entries copy in eight clocks)
Context 1
Context N
Logical Address & Context
Physical Address
Global Register Data Bus
Staging Data Buffer
Staging Write Addr
Physical Memory
Constants location available WRTR
Free list (pass Phys Address if Context Dirty)
Dealloc Counts
physical address to schedule for de-alloc
next physical address ready for allocate
Seq Constant Request
Logical address On the GlbRegBus when lsb are zero first word of write
Renaming Table for 1 Context Current/Last Physical Address per Logical Address
Reset Dirty per Logical Address (Only de-allocate if set)
This Context Dirty per Logical Address (If set don't allocate or de-allocate)
Context & Logical Address
Renaming table N-Contexts
Copy Last held above to Current Context on receipt of Set Constant for a new context (Hide loading behind Set State load - 16 clocks) all other Set States just write one entry to current state.

Figure 8: Constant management

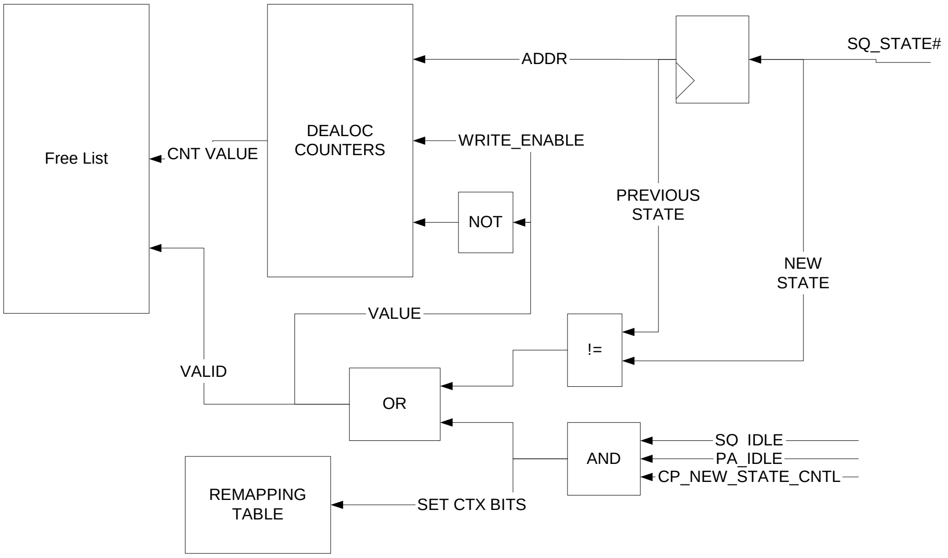


Figure 9: De-allocation mechanism for R400LE

5.3.3 *Dirty bits*

Two sets of dirty bits will be maintained per logical address. The first one will be set to zero on reset and set when the logical address is addressed. The second one will be set to zero whenever a new context is written and set for each address written while in this context. The reset dirty is not set, then writing to that logical address will not require de-allocation of whatever address stored in the renaming table. If it is set and the context dirty is not set, then the physical address store needs to be de-allocated and a new physical address is necessary to store the incoming data. If they are both set, then the data will be written into the physical address held in the renaming for the current logical address. No de-allocation or allocation takes place. This will happen when the driver does a set constant twice to the same logical address between context changes. NOTE: It is important to detect and prevent this, failure to do it will allow multiple writes to allocate all physical memory and thus hang because a context will not fit for rendering to start and thus free up space.

5.3.4 *Free List Block*

A free list block that would consist of a counter (called the IFC or Initial Free Counter) that would reset to zero and incremented every time a chunk of physical memory is used until they have all been used once. This counter would be checked each time a physical block is needed, and if the original ones have not been used up, us a new one, else check the free list for an available physical block address. The count is the physical address for when getting a chunk from the counter.

Storage of a free list big enough to store all physical block addresses.

Maintain three pointers for the free list that are reset to zero. The first one we will call write_ptr. This pointer will identify the next location to write the physical address of a block to be de-allocated. Note: we can never free more physical memory locations than we have. Once recording address the pointer will be incremented to walk the free list like a ring.

The second pointer will be called stop_ptr. The stop_ptr pointer will be advanced by the number of address chunks de-allocates when a context finishes. The address between the stop_ptr and write_ptr cannot be reused because they are still in use. But as soon as the context using then is dismissed the stop_ptr will be advanced.

The third pointer will be called read_ptr. This pointer will point will point to the next address that can be used for allocation as long as the read_ptr does not equal the stop_ptr and the IFC is at its maximum count.

5.3.5 *De-allocate Block*

This block will maintain a free physical address block count for each context. While in current context, a count shall be maintained specifying how many blocks were written into the free list at the write_ptr pointer. This count will be reset upon reset or when this context is active on the back and different than the previous context. It is actually a count of blocks in the previous context that will no longer be used. This count will be used to advance the write_ptr pointer to make available the set of physical blocks freed when the previous context was done. This allows the discard or de-allocation of any number of blocks in one clock.

5.3.6 *Operation of Incremental model*

The basic operation of the model would start with the write_ptr, stop_ptr, read_ptr pointers in the free list set to zero and the free list counter is set to zero. Also all the dirty bits and the previous context will be initialized to zero. When the first set constants happen, the reset dirty bit will not be set, so we will allocate a physical location from the free list counter because its not at the max value. The data will be written into physical address zero. Both the additional copy of the renaming table and the context zeros of the big renaming table will be updated for the logical address that was written by set start with physical address of 0. This process will be repeated for any logical address that are not dirty until the context changes. If a logical address is hit that has its dirty bits set while in the same context, both dirty bits would be set, so the new data will be over-written to the last physical address assigned for this logical address. When the first draw command of the context is detected, the previous context stored in the additional renaming table will be copied to the larger renaming table in the current (new) context location. Then the set constant logical address with be loaded with a new physical address during the copy and if the reset dirty was set, the physical address it replaced in the renaming table would be entered at the write_ptr pointer location on the free list and the write_ptr will be incremented. The de-allocation counter for the previous context (eight) will be incremented. This as set states come in for this context one of the following will happen:

1.) No dirty bits are set for the logical address being updated. A line will be allocated of the free-list counter or the free list at read_ptr pointer if read_ptr != to stop_ptr .
2.) Reset dirty set and Context dirty not set. A new physical address is allocated, the physical address in the renaming table is put on the free list at write_ptr and it is incremented along with the de-allocate counter for the last context.
3.) Context dirty is set then the data will be written into the physical address specified by the logical address.

This process will continue as long as set states arrive. This block will provide backpressure to the CP whenever he has not free list entries available (counter at max and stop_ptr == read_ptr). The command stream will keep a count of contexts of constants in use and prevent more than max constants contexts from being sent.

Whenever a draw packet arrives, the content of the re-mapping table is written to the correct re-mapping table for the context number. Also if the next context uses less constants than the current one all exceeding lines are moved to the free list to be de-allocated later. This happens in parallel with the writing of the re-mapping table to the correct memory.

Now preferable when the constant context leaves the last ALU clause it will be sent to this block and compared with the previous context that left. (Init to zero) If they differ than the older context will no longer be referenced and thus can be de-allocated in the physical memory. This is accomplished by adding the number of blocks freed this context to the stop_ptr pointer. This will make all the physical addresses used by this context available to the read_ptr allocate pointer for future allocation.

This device allows representation of multiple contexts of constants data with N copies of the logical address space. It also allows the second context to be represented as the first set plus some new additional data by just storing the delta's. It allows memory to be efficiently used and when the constants updates are small it can store multiple context. However, if the updates are large, less contexts will be stored and potentially performance will be degraded. Although it will still perform as well as a ring could in this case.

5.4 Constant Store Indexing

In order to do constant store indexing, the sequencer must be loaded first with the indexes (that come from the GPRs). There are 144 wires from the exit of the SP to the sequencer (9 bits pointers x 16 vertexes/clock). Since the data must pass thru the Shader pipe for the float to fixed conversion, there is a latency of 4 clocks (1 instruction)

between the time the sequencer is loaded and the time one can index into the constant store. The assembly will look like this

```
MOVA  R1.X,R2.X       // Loads the sequencer with the content of R2.X, also copies the content of R2.X into R1.X
NOP                   // latency of the float to fixed conversion
ADD   R3,R4,C0[R2.X]// Uses the state from the sequencer to add R4 to C0[R2.X] into R3
```

Note that we don't really care about what is in the brackets because we use the state from the MOVA instruction. R2.X is just written again for the sake of simplicity and coherency.

The storage needed in the sequencer in order to support this feature is 2*64*9 bits = 1152 bits.

5.5 Real Time Commands

The real time commands constants are written by the CP using the register mapped registers allocated for RT. It works is the same way than when dealing with regular constant loads BUT in this case the CP is not sending a logical address but rather a physical address and the reads are not passing thru the re-mapping table but are directly read from the memory. The boundary between the two zones is defined by the CONST_EO_RT control register. Similarly, for the fetch state, the boundary between the two zones is defined by the TSTATE_EO_RT control register.

5.6 Constant Waterfalling

In order to have a reasonable performance in the case of constant store indexing using the address register, we are going to have the possibility of using the physical memory port for read only. This way we can read 1 constant per clock and thus have a worst-case waterfall mode of 1 vertex per clock. There is a small synchronization issue related with this as we need for the SQ to make sure that the constants where actually written to memory (not only sent to the sequencer) before it can allow the first vector of pixels or vertices of the state to go thru the ALUs. To do so, the sequencer keeps 8 bits (one per render state) and sets the bits whenever the last render state is written to memory and clears the bit whenever a state is freed.

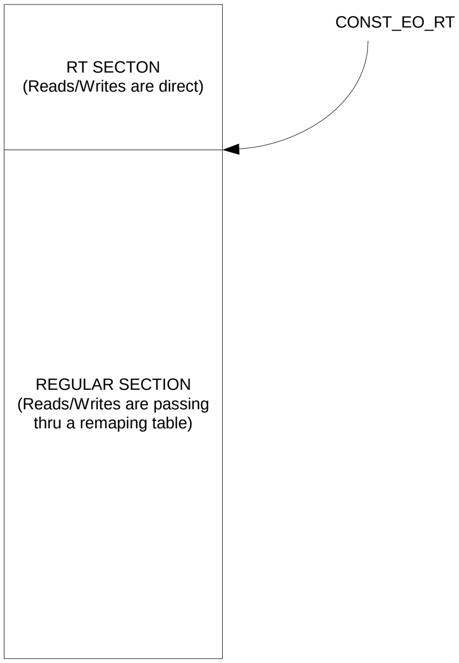


Figure 10: The instruction store

6. Looping and Branches

Loops and branches are planned to be supported and will have to be dealt with at the sequencer level. We plan on supporting constant loops and branches using a control program.

6.1 The controlling state.

The R400 controling state consists of:

Boolean[256:0]
Loop_count[7:0][31:0]
Loop_Start[7:0][31:0]
Loop_Step[7:0][31:0]

That is 256 Booleans and 32 loops.

We have a stack of 4 elements for nested calls of subroutines and 4 loop counters to allow for nested loops.

This state is available on a per shader program basis.

6.2 The Control Flow Program

Examples of control flow programs are located in the R400 programming guide document.

The basic model is as follows:

The render state defined the clause boundaries:
Vertex_shader_fetch[7:0][7:0]   // eight 8 bit pointers to the location where each clauses control program is located
Vertex_shader_alu[7:0][7:0]   // eight 8 bit pointers to the location where each clauses control program is located
Pixel_shader_fetch[7:0][7:0]   // eight 8 bit pointers to the location where each clauses control program is located
Pixel_shader_alu[7:0][7:0]   // eight 8 bit pointers to the location where each clauses control program is located

A pointer value of FF means that the clause doesn't contain any instructions.

The control program for a given clause is executed to completion before moving to another clause, (with the exception of the pick two nature of the alu execution). The control program is the only program aware of the clause boundaries.

The control program has nine basic instructions:

Execute
Conditional_execute
Conditional_Execute_Predicates
Conditional_jump
Conditionnal_Call
Return
Loop_start
Loop_end
NOP

Execute, causes the specified number of instructions in instruction store to be executed.
Conditional_execute checks a condition first, and if true, causes the specified number of instructions in instruction store to be executed.
Loop_start resets the corresponding loop counter to the start value on the first pass after it checks for the end condition and if met jumps over to a specified address.
Loop_end increments (decrements?) the loop counter and jumps back the specified number of instructions.
Conditionnal_Call jumps to an address and pushes the IP counter on the stack if the condition is met. On the return instruction, the IP is popped from the stack.

Conditional_execute_Predicates executes a block of instructions if all bits in the predicate vectors meet the condition.
Conditional_jumps jumps to an address if the condition is met.
NOP is a regular NOP

NOTE THAT ALL JUMPS MUST JUMP TO EVEN CFP ADDRESSES since there are two control flow instructions per memory line. Thus the compiler must insert NOPs where needed to align the jumps on even CFP addresses.

Also if the jump is logically bigger than pshader_cntl_size (or vshader_cntl_size) we break the program (clause) and set the debug registers. If an execute or conditional_execute is lower than cntl_size or bigger than size we also break the program (clause) and set the debug registers.

We have to fit instructions into 48 bits in order to be able to put two control flow instruction per line in the instruction store.

A value of 1 in the Addressing means that the address specified in the Exec Address field (or in the jump address field) is an ABSOLUTE address. If the addressing field is cleared (should be the default) then the address is relative to the base of the current shader program.

Note that whenever a field is marked as RESERVED, it is assumed that all the bits of the field are cleared (0).

Execute					
47	46… 42	41	40 … 24	23 … 12	11 … 0
Addressing	00001	Last	RESERVED	Instruction count	Exec Address

Execute up to 4k instructions at the specified address in the instruction memory. If Last is set, this is the last group of instructions of the clause.

NOP			
47	46 … 42	41	40 … 0
Addressing	00010	Last	RESERVED

This is a regular NOP. If Last is set, this is the last instruction of the clause.

Conditional_Execute								
47	46 … 42	41	40	~~40~~ 39 … ~~33~~32	~~32~~31	~~31~~ 30 … 24	23 … 12	11 … 0
Addressing	00011	Last	RESERVED	Boolean address	Condition	RESERVED	Instruction count	Exec Address

If the specified Boolean (8 bits can address 256 Booleans) meets the specified condition then execute the specified instructions (up to 4k instructions). If Last is set, then if the condition is met, this is the last group of instructions to be executed in the clause. If the condition is not met, we go on to the next control flow instruction.

Conditional_Execute_Predicates								
47	46 … 42	41	~~40~~ 40 … ~~35~~34	~~34~~ 33 … ~~33~~32	~~32~~31	~~31~~ 30 … 24	23 … 12	11 … 0
Addressing	00100	Last	RESERVED	Predicate vector	Condition	RESERVED	Instruction count	Exec Address

Check the AND/OR of all current predicate bits. If AND/OR matches the condition execute the specified number of instructions. We need to AND/OR this with the kill mask in order not to consider the pixels that aren't valid. If Last is set, then if the condition is met, this is the last group of instructions to be executed in the clause. If the condition is not met, we go on to the next control flow instruction.

Loop_Start				
47	46 … 42	41 … 17	16 … 12	11 … 0

Addressing	00101	RESERVED	loop ID	Jump address

Loop Start. Compares the loop iterator with the end value. If loop condition not met jump to the address. Forward jump only. Also computes the index value. The loop id must match between the start to end, and also indicates which control flow constants should be used with the loop.

Loop_End				
47	46 … 42	41 … 17	16 … 12	11 … 0
Addressing	00110	RESERVED	loop ID	start address

Loop end. Increments the counter by one, compares the loop count with the end value. If loop condition met, continue, else, jump BACK to the start of the loop.

The way this is described does not prevent nested loops, and the inclusion of the loop id make this easy to do.

Conditionnal_Call						
47	46 … 42	41 … ~~35~~34	~~34~~ 33 … ~~33~~32	31~~2~~	~~31~~ 30 … 12	11 … 0
Addressing	00111	RESERVED	Predicate vector	Condition	RESERVED	Jump address

If the condition is met, jumps to the specified address and pushes the control flow program counter on the stack.

Return		
47	46 … 42	41 … 0
Addressing	01000	RESERVED

Pops the topmost address from the stack and jumps to that address. If nothing is on the stack, the program will just continue to the next instruction.

Conditionnal_Jump							
47	46 … 42	41 … 40~~1~~	~~40~~ 39 … ~~33~~32	~~32~~31	~~31~~30	~~30~~ 29 … 12	11 … 0
Addressing	01001	RESERVED	Boolean address	Condition	FW only	RESERVED	Jump address

If condition met, jumps to the address. FORWARD jump only allowed if bit 31 set. Bit 31 is only an optimization for the compiler and should NOT be exposed to the API.

To prevent infinite loops, we will keep 9 bits loop iterators instead of 8 (we are only able to loop 256 times). If the counter goes higher than 255 then the loop_end or the loop_start instruction is going to break the loop and set the debug GPRs.

6.3 Data dependant predicate instructions

Data dependant conditionals will be supported in the R400. The only way we plan to support those is by supporting three vector/scalar predicate operations of the form:

PRED_SETE_# - similar to SETE except that the result is 'exported' to the sequencer.
PRED_SETNE_# - similar to SETNE except that the result is 'exported' to the sequencer.
PRED_SETGT_# - similar to SETGT except that the result is 'exported' to the sequencer
PRED_SETGTE_# - similar to SETGTE except that the result is 'exported' to the sequencer

For the scalar operations only we will also support the two following instructions:
PRED_SETE0_# – SETE0

PRED_SETE1_# – SETE1

The export is a single bit - 1 or 0 that is sent using the same data path as the MOVA instruction.   The sequencer will maintain 4 sets of 64 bit predicate vectors (in fact 8 sets because we interleave two programs but only 4 will be exposed) and use it to control the write masking. This predicate is not maintained across clause boundaries. The # sign is used to specify which predicate set you want to use 0 thru 3.

Then we have two conditional execute bits. The first bit is a conditional execute "on" bit and the second bit tells us if we execute on 1 or 0. For example, the instruction:

P0_ADD_# R0,R1,R2

Is only going to write the result of the ADD into those GPRs whose predicate bit is 0. Alternatively, P1_ADD_# would only write the results to the GPRs whose predicate bit is set. The use of the P0 or P1 without precharging the sequencer with a PRED instruction is undefined.

{Issue: do we have to have a NOP between PRED and the first instruction that uses a predicate?}

6.4 HW Detection of PV,PS

Because of the control program, the compiler cannot detect statically dependant instructions. In the case of non-masked writes and subsequent reads the sequencer will insert uses of PV,PS as needed. This will be done by comparing the read address and the write address of consecutive instructions. For masked writes, the sequencer will insert NOPs wherever there is a dependant read/write.

The sequencer will also have to insert NOPs between PRED_SET and MOVA instructions and their uses.

6.5 Register file indexing

Because we can have loops in fetch clause, we need to be able to index into the register file in order to retrieve the data created in a fetch clause loop and use it into an ALU clause. The instruction will include the base address for register indexing and the instruction will contain these controls:

Bit7	Bit 6	
0	0	'absolute register'
0	1	'relative register'
1	0	'previous vector'
1	1	'previous scalar'

In the case of an absolute register we just take the address as is. In the case of a relative register read we take the base address and we add to it the loop_index and this becomes our new address that we give to the shader pipe.

The sequencer is going to keep a loop index computed as such:

Index = Loop_iterator*Loop_step + Loop_start.

We loop until loop_iterator = loop_count. Loop_step is a signed value [-128…127]. The computed index value is a 10 bit counter that is also signed. Its real range is [-256,256]. The tenth bit is only there so that we can provide an out of range value to the "indexing logic" so that it knows when the provided index is out of range and thus can make the necessary arrangements.

6.6 Predicated Instruction support for Texture clauses

For texture clauses, we support the following optimization: we keep 1 bit (thus 4 bits for the four predicate vectors) per predicate vector in the reservation stations. A value of 1 means that one ore more elements in the vector have a value of one (thus we have to do the texture fetches for the whole vector). A value of 0 means that no elements in the vector have his predicate bit set and we can thus skip over the texture fetch. **We have to make sure the invalid pixels aren't considered with this optimization.**

6.7 Debugging the Shaders

In order to be able to debug the pixel/vertex shaders efficiently, we provide 2 methods.

6.7.1 *Method 1: Debugging registers*

Current plans are to expose 2 debugging, or error notification, registers:
1. address register where the first error occurred
2. count of the number of errors

The sequencer will detect the following groups of errors:
- count overflow
- constant indexing overflow
- register indexing overflow

Compiler recognizable errors:
- jump errors
    relative jump address > size of the control flow program
- call stack
    call with stack full
    return with stack empty

A jump error will always cause the program to break. In this case, a break means that a clause will halt execution, but allowing further clauses to be executed.

With all the other errors, program can continue to run, potentially to worst-case limits. The program will only break if the DB_PROB_BREAK register is set.

If indexing outside of the constant or the register range, causing an overflow error, the hardware is specified to return the value with an index of 0. This could be exploited to generate error tokens, by reserving and initializing the 0th register (or constant) for errors.

{ISSUE : Interrupt to the driver or not?}

6.7.2 *Method 2: Exporting the values in the GPRs (12)*

The sequencer will have a count register and an address register for this mode and 3 bits per clause specifying the execution mode for each clause. The modes can be :

1) Normal
2) Debug Kill
3) Debug Addr + Count

Under the normal mode execution follows the normal course. Under the kill mode, all control flow instructions are executed but all normal shader instructions of the clause are replaced by NOPs. Only debug_export instructions of clause 7 will be executed under the debug kill setting. Under the other mode, normal execution is done until we reach an address specified by the address register and instruction count (useful for loops) specified by the count register. After we have hit the instruction n times (n=count) we switch the clause to the kill mode.

Under the debug mode (debug kill OR debug Addr + count), it is assumed that clause 7 is always exporting 12 debug vectors and that all other exports to the SX block (position, color, z, ect) will been turned off (changed into NOPs) by the sequencer (even if they occur before the address stated by the ADDR debug register).

7. Pixel Kill Mask

A vector of 64 bits is kept by the sequencer per group of pixels/vertices. Its purpose is to optimize the texture fetch requests and allow the shader pipe to kill pixels using the following instructions:

MASK_SETE
MASK_SETNE
MASK_SETGT

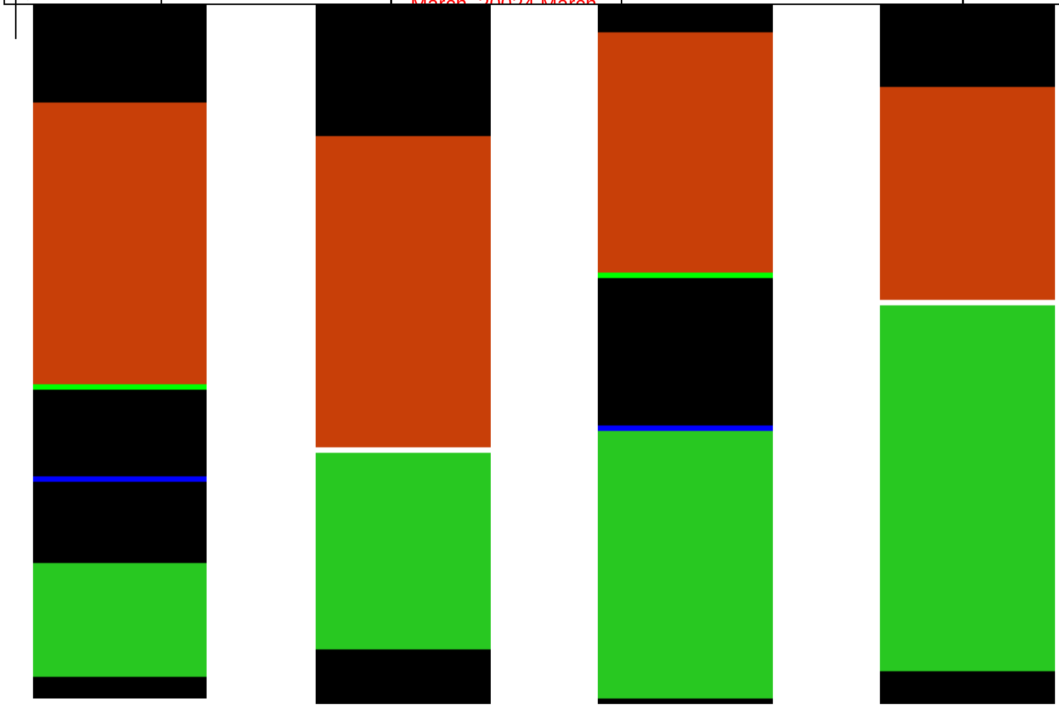
MASK_SETGTE

8. Multipass vertex shaders (HOS)

Multipass vertex shaders are able to export from the 6 last clauses but to memory ONLY.

9. Register file allocation

The register file allocation for vertices and pixels can either be static or dynamic. In both cases, the register file in managed using two round robins (one for pixels and one for vertices). In the dynamic case the boundary between pixels and vertices is allowed to move, in the static case it is fixed to 128-VERTEX_REG_SIZE for vertices and PIXEL_REG_SIZE for pixels.



Above is an example of how the algorithm works. Vertices come in from top to bottom; pixels come in from bottom to top. Vertices are in orange and pixels in green. The blue line is the tail of the vertices and the green line is the tail of the pixels. Thus anything between the two lines is shared. When pixels meets vertices the line turns white and the boundary is static until both vertices and pixels share the same “unallocated bubble”. Then the boundary is allowed to move again. The numbering of the GPRs starts from the bottom of the picture at index 0 and goes up to the top at index 127.

10. Fetch Arbitration

The fetch arbitration logic chooses one of the 8 potentially pending fetch clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute. Once chosen, the clause state machine will send one 2x2 fetch per clock (or 4 fetches in one clock every 4 clocks) until all the fetch instructions of the clause are sent. This means that there cannot be any dependencies between two fetches of the same clause.

The arbitrator will not wait for the fetches to return prior to selecting another clause for execution. The fetch pipe will be able to handle up to X(?) in flight fetches and thus there can be a fair number of active clauses waiting for their fetch return data.

11. ALU Arbitration

ALU arbitration proceeds in almost the same way than fetch arbitration. The ALU arbitration logic chooses one of the 8 potentially pending ALU clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute. There are two ALU arbiters, one for the even clocks and one for the odd clocks. For example, here is the sequencing of two interleaved ALU clauses (E and O stands for Even and Odd sets of 4 clocks):

Einst0 Oinst0 Einst1 Oinst1 Einst2 Oinst2 Einst0 Oinst3 Einst1 Oinst4 Einst2 Oinst0…
Proceeding this way hides the latency of 8 clocks of the ALUs. Also note that the interleaving also occurs across clause boundaries.

12. Handling Stalls

When the output file is full, the sequencer prevents the ALU arbitration logic from selecting the last clause (this way nothing can exit the shader pipe until there is place in the output file. If the packet is a vertex packet and the position buffer is full (POS_FULL) then the sequencer also prevents a thread from entering the exporting clause (3?). The sequencer will set the OUT_FILE_FULL signal n clocks before the output file is actually full and thus the ALU arbiter will be able read this signal and act accordingly by not preventing exporting clauses to proceed.

13. Content of the reservation station FIFOs

The reservation FIFOs contain the state of the vector of pixels and vertices. We have two sets of those: one for pixels, and one for vertices. They contain 3 bits of Render State 7 bits for the base address of the GPRs, some bits for LOD correction and coverage mask information in order to fetch fetch for only valid pixels, the quad address.

14. The Output File

The output file is where pixels are put before they go to the RBs. The write BW to this store is 256 bits/clock. Just before this output file are staging registers with write BW 512 bits/clock and read BW 256 bits/clock. The staging registers are 4x128 (and there are 16 of those on the whole chip).

15. IJ Format

The IJ information sent by the PA is of this format on a per quad basis:

We have a vector of IJ's (one IJ per pixel at the centroid of the fragment or at the center of the pixel depending on the mode bit). The interpolation is done at a different precision across the 2x2. The upper left pixel's parameters are always interpolated at full 20x24 mantissa precision. Then the result of the interpolation along with the difference in IJ in reduced precision is used to interpolate the parameter for the other three pixels of the 2x2. Here is how we do it:

Assuming P0 is the interpolated parameter at Pixel 0 having the barycentric coordinates I(0), J(0) and so on for P1,P2 and P3. Also assuming that A is the parameter value at V0 (interpolated with I), B is the parameter value at V1 (interpolated with J) and C is the parameter value at V2 (interpolated with (1-I-J).

$$\Delta 01I = I(1) - I(0)$$
$$\Delta 01J = J(1) - J(0)$$
$$\Delta 02I = I(2) - I(0)$$
$$\Delta 02J = J(2) - J(0)$$
$$\Delta 03I = I(3) - I(0)$$
$$\Delta 03J = J(3) - J(0)$$

P0	P1
P2	P3

$$P0 = C + I(0) * (A - C) + J(0) * (B - C)$$
$$P1 = P0 + \Delta 01I * (A - C) + \Delta 01J * (B - C)$$
$$P2 = P0 + \Delta 02I * (A - C) + \Delta 02J * (B - C)$$
$$P3 = P0 + \Delta 03I * (A - C) + \Delta 03J * (B - C)$$

P0 is computed at 20x24 mantissa precision and P1 to P3 are computed at 8X24 mantissa precision. So far no visual degradation of the image was seen using this scheme.

Multiplies (Full Precision): 2
Multiplies (Reduced precision): 6
Subtracts 19x24 (Parameters): 2

Adds: 8

FORMAT OF P0's IJ : Mantissa 20 Exp 4 for I + Sign
Mantissa 20 Exp 4 for J + Sign

FORMAT of Deltas (x3):Mantissa 8 Exp 4 for I + Sign
Mantissa 8 Exp 4 for J + Sign

Total number of bits : 20*2 + 8*6 + 4*8 + 4*2 = 128

All numbers are kept using the un-normalized floating point convention: if exponent is different than 0 the number is normalized if not, then the number is un-normalized. The maximum range for the IJs (Full precision) is +/- 63 and the range for the Deltas is +/- 127.

15.1 Interpolation of constant attributes

Because of the floating point imprecision, we need to take special provisions if all the interpolated terms are the same or if two of the barycentric coordinates are the same.

We start with the premise that if A = B and B = C and C = A, then P0,1,2,3 = A. Since one or more of the IJ terms may be zero, so we extend this to:

```
if (A=B and B=C and C=A)
   P0,1,2,3 = A;
else if ((I = 0) or (J = 0)) and
          ((J = 0) or (1-I-J = 0)) and
          ((1-J-I = 0) or (I = 0))) {
                if(I != 0) {
                    P0 = A;
                } else if(J != 0) {
                    P0 = B;
                } else {
                    P0 = C;
                }
             //rest of the quad interpolated normally
}
else
{
             normal interpolation
}
```

16. Staging Registers

In order for the reuse of the vertices to be 14, the sequencer will have to re-order the data sent IN ORDER by the VGT for it to be aligned with the parameter cache memory arrangement. Given the following group of vertices sent by the VGT:

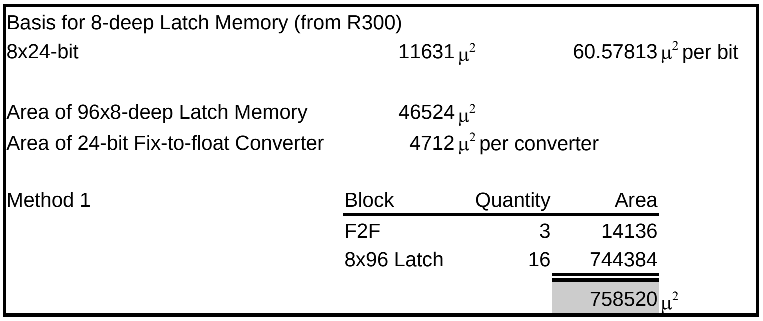
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 || 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 || 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 || 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63

The sequencer will re-arrange them in this fashion:

0 1 2 3 16 17 18 19 32 33 34 35 48 49 50 51 || 4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55 || 8 9 10 11 24 25 26 27 40 41 42 43 56 57 58 59 || 12 13 14 15 28 29 30 31 44 45 46 47 60 61 62 63

The || markers show the SP divisions. In the event a shader pipe is broken, the VGT will send padding to account for the missing pipe. For example, if SP1 is broken, vertices 4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55 will still be sent by the VGT to the SQ **BUT** will not be processed by the SP and thus should be considered invalid (by the SU and VGT).

The most straightforward, *non-compressed* interface method would be to convert, in the VGT, the data to 32-bit floating point prior to transmission to the VSISRs. In this scenario, the data would be transmitted to (and stored in) the VSISRs in full 32-bit floating point. This method requires three 24-bit fixed-to-float converters in the VGT. Unfortunately, it also requires and additional 3,072 bits of storage across the VSISRs. This interface is illustrated in Figure 12~~Figure 12Figure 12~~. The area of the fixed-to-float converters and the VSISRs for this method is roughly estimated as 0.759sqmm using the R300 process. The gate count estimate is shown in Figure 11~~Figure 11Figure 11~~.



Basis for 8-deep Latch Memory (from R300)				
8x24-bit	11631 μ^2		60.57813 μ^2 per bit	
Area of 96x8-deep Latch Memory	46524 μ^2			
Area of 24-bit Fix-to-float Converter	4712 μ^2 per converter			
Method 1	Block	Quantity	Area	
	F2F	3	14136	
	8x96 Latch	16	744384	
			758520	μ^2

Figure 11:Area Estimate for VGT to Shader Interface

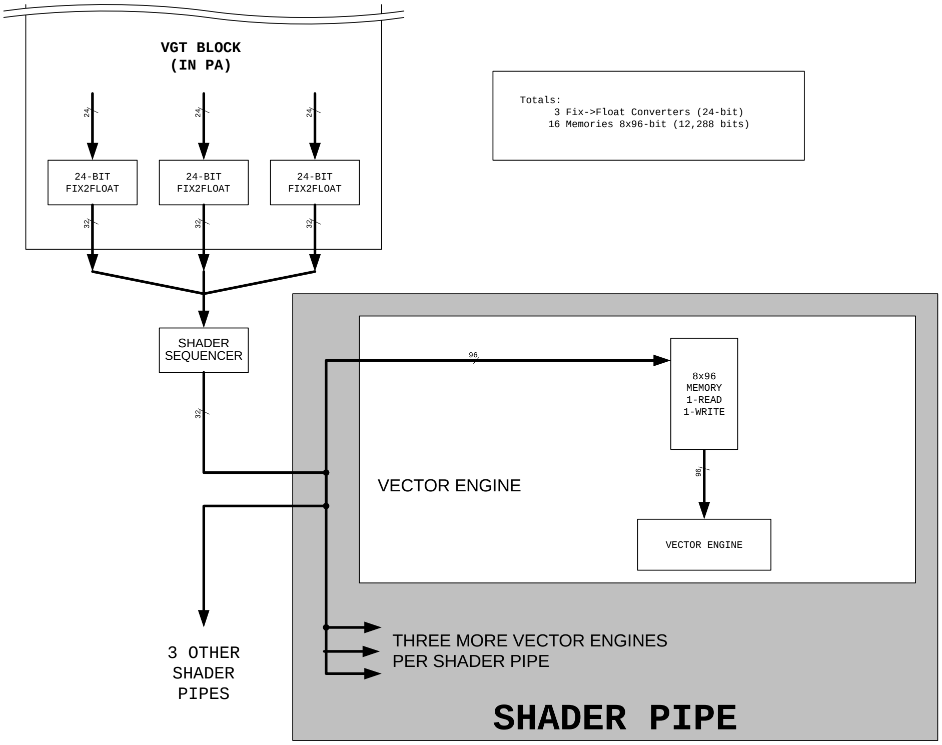


Figure 12:VGT to Shader Interface

17. The parameter cache

The parameter cache is where the vertex shaders export their data. It consists of 16 128x128 memories (1R/1W). The reuse engine will make it so that all vertexes of a given primitive will hit different memories. The allocation method for these memories is a simple round robin. The parameter cache pointers are mapped in the following way: 4MSBs are the memory number and the 7 LSBs are the address within this memory.

MEMORY NUMBER	ADDRESS
4 bits	7 bits

The PA generates the parameter cache addresses as the positions come from the SQ. All it needs to do is keep a Current_Location pointer (7 bits only) and as the positions comes increment the memory number. When the memory number field wraps around, the PA increments the Current_Location by VS_EXPORT_COUNT_7 (a snooped register from the SQ). As an example, say the memories are all empty to begin with and the vertex shader is exporting 8 parameters per vertex (VS_EXPORT_COUNT_7 = 8). The first position received is going to have the PC address 00000000000 the second one 00010000000, third one 00100000000 and so on up to 11110000000. Then the next position received (the 17th) is going to have the address 00000001000, the 18th 00010001000, the 19th 00100001000 and so on. The Current_location is NEVER reset BUT on chip resets. The only thing to be careful about is that if the SX doesn't send you a full group of positions (<64) then you need to fill the address space so that the next group starts correctly aligned (for example if you receive only 33 positions then you need to add 2*VS_EXPORT_COUNT_7to Current_Location and reset the memory count to 0 before the next vector begins).

18. Vertex position exporting

On clause 3 the vertex shader can export to the PA both the vertex position and the point sprite. It can also do so at clause 7 if not done at clause 3. The storage needed to perform the position export is at least 64x128 memories for the position and 64x32 memories for the sprite size. It is going to be taken in the pixel output fifo from the SX blocks. The clause where the position export occurs is specified by the EXPORT_LATE register. If turned on, it means that the export is going to occur at ALU clause 7 if unset position export occurs at clause 3.

19. Exporting Arbitration

Here are the rules for co-issuing exporting ALU clauses.

1) Position exports and position exports cannot be co-issued.

All other types of exports can be co-issued as long as there is place in the receiving buffer.

~~{ISSUE: Do we move the parameter caches to the SX?}~~

Formatted: Bullets and Numbering

20. Exporting Rules

20.1 Parameter caches exports

We support masking and out of order exports to the parameter caches. So one can export multiple times to the same PC line using different masks.

Formatted: Bullets and Numbering

20.2 Memory exports

Memory exports don't support masking. However, you can export out of order to memory locations.

Formatted: Bullets and Numbering

20.3 Position exports

Position exports have to be done IN ORDER and don't support masking.

Formatted: Bullets and Numbering

~~20.~~21. Export Types

The export type (or the location where the data should be put) is specified using the destination address field in the ALU instruction. Here is a list of all possible export modes:

Formatted: Bullets and Numbering

~~20.1~~21.1 Vertex Shading

0:15	- 16 parameter cache
16:31	- Empty (Reserved?)
32:43	- 12 vertex exports to the frame buffer and index
44:47	- Empty
48:59	- 12 debug export (interpret as normal vertex export)
60	- export addressing mode
61	- Empty
62	- position
63	- sprite size export that goes with position export (point_h,point_w,edgeflag,misc)

Formatted: Bullets and Numbering

~~20.2~~21.2 Pixel Shading

0	- Color for buffer 0 (primary)
1	- Color for buffer 1

2	- Color for buffer 2
3	- Color for buffer 3
4:7	- Empty
8	- Buffer 0 Color/Fog (primary)
9	- Buffer 1 Color/Fog
10	- Buffer 2 Color/Fog
11	- Buffer 3 Color/Fog
12:15	- Empty
16:31	- Empty (Reserved?)
32:43	- 12 exports for multipass pixel shaders.
44:47	- Empty
48:59	- 12 debug exports (interpret as normal pixel export)
60	- export addressing mode
61:62	- Empty
63	- Z for primary buffer (Z exported to 'alpha' component)

Formatted: Bullets and Numbering

~~21.~~22. Special Interpolation modes

~~21.1~~22.1 Real time commands

We are unable to use the parameter memory since there is no way for a command stream to write into it. Instead we need to add three 16x128 memories (one for each of three vertices x 16 interpolants). These will be mapped onto the register bus and written by type 0 packets, and output to the the parameter busses (the sequencer and/or PA need to be able to address the realtime parameter memory as well as the regular parameter store. For higher performance we should be able able to view them as two banks of 16 and do double buffering allowing one to be loaded, while the other is rasterized with. Most overlay shaders will need 2 or 4 scalar coordinates, one option might be to restrict the memory to 16x64 or 32x64 allowing only two interpolated scalars per cycle, the only problem I see with this is, if we view support for 16 vector-4 interpolants important (true only if we map Microsoft's high priority stream to the realtime stream), then the PA/sequencer need to support a realtime-specific mode where we need to address 32 vectors of parameters instead of 16. This mode is triggered by the primitive type: REAL TIME. The actual memories are in the in the SX blocks. The parameter data memories are hooked on the RBBM bus and are loaded by the CP using register mapped memory.

Formatted: Bullets and Numbering

~~21.2~~22.2 Sprites/ XY screen coordinates/ FB information

When working with sprites, one may want to overwrite the parameter 0 with SC generated data. Also, XY screen coordinates may be needed in the shader program. This functionality is controlled by the gen_I0 register (in SQ) in conjunction with the SND_XY register (in SC). Also it is possible to send the faceness information (for OGL front/back special operations) to the shader using the same control register. Here is a list of all the modes and how they interact together:

Gen_st is a bit taken from the interface between the SC and the SQ. This is the MSB of the primitive type. If the bit is set, it means we are dealing with Point AA, Line AA or sprite and in this case the vertex values are going to generated between 0 and 1.

Param_Gen_I0 disable, snd_xy disable, no gen_st – I0 = No modification
Param_Gen_I0 disable, snd_xy disable, gen_st – I0 = No modification
Param_Gen_I0 disable, snd_xy enable, no gen_st – I0 = No modification
Param_Gen_I0 disable, snd_xy enable, gen_st – I0 = No modification
Param_Gen_I0 enable, snd_xy disable, no gen_st – I0 = garbage, garbage, garbage, faceness
Param_Gen_I0 enable, snd_xy disable, gen_st – I0 = garbage, garbage, s, t
Param_Gen_I0 enable, snd_xy enable, no gen_st – I0 = screen x, screen y, garbage, faceness
Param_Gen_I0 enable, snd_xy enable, gen_st – I0 = screen x, screen y, s, t

Formatted: Bullets and Numbering

~~21.3~~22.3 Auto generated counters

In the cases we are dealing with multipass shaders, the sequencer is going to generate a vector count to be able to both use this count to write the 1st pass data to memory and then use the count to retrieve the data on the 2nd pass. The count is always generated in the same way but it is passed to the shader in a slightly different way depending on the shader type (pixel or vertex). This is toggled on and off using the GEN_INDEX register. The sequencer is going to keep two counters, one for pixels and one for vertices. Every time a full vector of vertices or pixels is written to the GPRs the counter is incremented. Every time a state change is detected, the corresponding counter is reset. While there is only one count broadcast to the GPRs, the LSB are hardwired to specific values making the index different for all elements in the vector.

~~21.3.1~~22.3.1 *Vertex shaders*

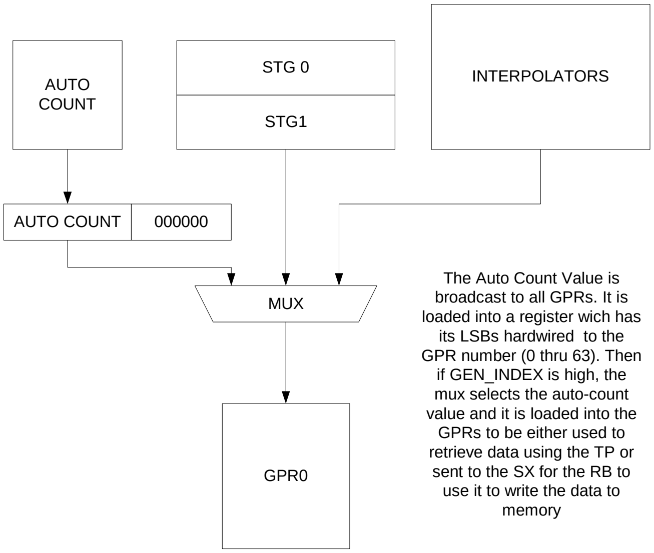
Formatted: Bullets and Numbering

In the case of vertex shaders, if GEN_INDEX is set, the data will be put into the x field of the third register (it means that the compiler must allocate 3 GPRs in all multipass vertex shader modes).

~~21.3.2~~22.3.2 *Pixel shaders*

Formatted: Bullets and Numbering

In the case of pixel shaders, if GEN_INDEX is set and Param_Gen_I0 is enabled, the data will be put in the x field of the 2nd register (R1.x), else if GEN_INDEX is set the data will be put into the x field of the 1st register (R0.x).



The Auto Count Value is broadcast to all GPRs. It is loaded into a register wich has its LSBs hardwired to the GPR number (0 thru 63). Then if GEN_INDEX is high, the mux selects the auto-count value and it is loaded into the GPRs to be either used to retrieve data using the TP or sent to the SX for the RB to use it to write the data to memory

Figure 13: GPR input mux Control

~~22.~~23. State management

Formatted: Bullets and Numbering

Every clock, the sequencer will report to the CP the oldest states still in the pipe. These are the states of the programs as they enter the last ALU clause.

~~22.1~~23.1 Parameter cache synchronization

Formatted: Bullets and Numbering

In order for the sequencer not to begin a group of pixels before the associated group of vertices has finished, the sequencer will keep a 6 bit count per state (for a total of 8 counters). These counters are initialized to 0 and every time a vertex shader exports its data TO THE PARAMETER CACHE, the corresponding pointer is incremented. When the SC sends a new vector of pixels with the SC_SQ_new_vector bit asserted, the sequencer will first check if the count is greater than 0 before accepting the transmission (it will in fact accept the transmission but then lower its ready to receive). Then the sequencer waits for the count to go to one and decrements it. The sequencer can then issue the group of pixels to the interpolators. Every time the state changes, the new state counter is initialized to 0.

~~23.~~24. XY Address imports

Formatted: Bullets and Numbering

The SC will be able to send the XY addresses to the GPRs. It does so by interleaving the writes of the IJs (to the IJ buffer) with XY writes (to the XY buffer). Then when writing the data to the GPRs, the sequencer is going to interpolate the IJ data or pass the XY data thru a Fix→float converter and expander and write the converted values to the GPRs. The Xys are currently SCREEN SPACE COORDINATES. The values in the XY buffers will wrap. See section 22.2~~21.2~~ for details on how to control the interpolation in this mode.

~~23.1~~24.1 Vertex indexes imports

Formatted: Bullets and Numbering

In order to import vertex indexes, we have 16 8x96 staging registers. These are loaded one line at a time by the VGT block (96 bits). They are loaded in floating point format and can be transferred in 4 or 8 clocks to the GPRs.

~~24.~~25. Registers

Formatted: Bullets and Numbering

~~24.1~~25.1 Control

REG_DYNAMIC	Dynamic allocation (pixel/vertex) of the register file on or off.
REG_SIZE_PIX	Size of the register file's pixel portion (minimal size when dynamic allocation turned on)
REG_SIZE_VTX	Size of the register file's vertex portion (minimal size when dynamic allocation turned on)
ARBITRATION_POLICY	policy of the arbitration between vertexes and pixels
INST_STORE_ALLOC	interleaved, separate
INST_BASE_VTX	start point for the vertex instruction store (RT always ends at vertex_base and Begins at 0)
INST_BASE_PIX	start point for the pixel shader instruction store
ONE_THREAD	debug state register. Only allows one program at a time into the GPRs
ONE_ALU	debug state register. Only allows one ALU program at a time to be executed (instead of 2)
INSTRUCTION	This is where the CP puts the base address of the instruction writes and type (auto-incremented on reads/writes) Register mapped
CONSTANTS	512*4 ALU constants + 32*6 Texture state 32 bits registers (logically mapped)
CONSTANTS_RT	256*4 ALU constants + 32*6 texture states? (physically mapped)
CONSTANT_EO_RT	This is the size of the space reserved for real time in the constant store (from 0 to CONSTANT_EO_RT). The re-mapping table operates on the rest of the memory
TSTATE_EO_RT	This is the size of the space reserved for real time in the fetch state store (from 0 to TSTATE_EO_RT). The re-mapping table operates on the rest of the memory
EXPORT_LATE	Controls whether or not we are exporting position from clause 3. If set, position exports occur at clause 7.

~~24.2~~25.2 Context

Formatted: Bullets and Numbering

VS_FETCH_{0…7}	eight 8 bit pointers to the location where each clauses control program is located
VS_ALU_{0…7}	eight 8 bit pointers to the location where each clauses control program is located
PS_FETCH_{0…7}	eight 8 bit pointers to the location where each clauses control program is located
PS_ALU_{0…7}	eight 8 bit pointers to the location where each clauses control program is located
PS_BASE	base pointer for the pixel shader in the instruction store
VS_BASE	base pointer for the vertex shader in the instruction store
VS_CF_SIZE	size of the vertex shader (# of instructions in control program/2)
PS_CF_SIZE	size of the pixel shader (# of instructions in control program/2)
PS_SIZE	size of the pixel shader (cntl+instructions)
VS_SIZE	size of the vertex shader (cntl+instructions)
PS_NUM_REG	number of GPRs to allocate for pixel shader programs
VS_NUM_REG	number of GPRs to allocate for vertex shader programs
PARAM_SHADE	One 16 bit register specifying which parameters are to be gouraud shaded (0 = flat, 1 = gouraud)
PROVO_VERT	0 : vertex 0, 1: vertex 1, 2: vertex 2, 3: Last vertex of the primitive

PARAM_WRAP	64 bits: for which parameters (and channels (xyzw)) do we do the cyl wrapping (0=linear, 1=cylindrical).
PS_EXPORT_MODE	0xxxx : Normal mode 1xxxx : Multipass mode If normal, bbbz where bbb is how many colors (0-4) and z is export z or not If multipass 1-12 exports for color.
~~VS_EXPORT_MASK~~	~~which of the last 6 ALU clauses is exporting (multipass only)~~
VS_EXPORT_MODE	0: position (1 vector), 1: position (2 vectors), 3:multipass
VS_EXPORT _COUNT_{0…6}	Six 4 bit counters representing the # of interpolated parameters exported in clause 7 (located in VS_EXPORT_COUNT_6) OR # of exported vectors to memory per clause in multipass mode (per clause)
PARAM_GEN_I0	Do we overwrite or not the parameter 0 with XY data and generated T and S values
GEN_INDEX	Auto generates an address from 0 to XX. Puts the results into R0-1 for pixel shaders and R2 for vertex shaders
CONST_BASE_VTX (9 bits)	Logical Base address for the constants of the Vertex shader
CONST_BASE_PIX (9 bits)	Logical Base address for the constants of the Pixel shader
CONST_SIZE_PIX (8 bits)	Size of the logical constant store for pixel shaders
CONST_SIZE_VTX (8 bits)	Size of the logical constant store for vertex shaders
INST_PRED_OPTIMIZE	Turns on the predicate bit optimization (if of, conditional_execute_predicates is always executed).
CF_BOOLEANS	256 boolean bits
CF_LOOP_COUNT	32x8 bit counters (number of times we traverse the loop)
CF_LOOP_START	32x8 bit counters (init value used in index computation)
CF_LOOP_STEP	32x8 bit counters (step value used in index computation)

Formatted: Bullets and Numbering

~~25.~~26. DEBUG Registers

~~25.1~~26.1 Context

DB_PROB_ADDR	instruction address where the first problem occurred
DB_PROB_COUNT	number of problems encountered during the execution of the program
DB_PROB_BREAK	break the clause if an error is found.
DB_INST_COUNT	instruction counter for debug method 2
DB_BREAK_ADDR	break address for method number 2
DB_CLAUSE _MODE_ALU_{0…7}	clause mode for debug method 2 (0: normal, 1: addr, 2: kill)
DB_CLAUSE _MODE_FETCH_{0…7}	clause mode for debug method 2 (0: normal, 1: addr, 2: kill)

Formatted: Bullets and Numbering

~~25.2~~26.2 Control

DB_ALUCST_MEMSIZE	Size of the physical ALU constant memory
DB_TSTATE_MEMSIZE	Size of the physical texture state memory

Formatted: Bullets and Numbering

~~26.~~27. Interfaces

~~26.1~~27.1 External Interfaces

Whenever an x is used, it means that the bus is broadcast to all units of the same name. For example, if a bus is named SQ→SPx it means that SQ is going to broadcast the same information to all SP instances.

Formatted: Bullets and Numbering

27.2 SC to SP Interfaces

27.2.1 SC_SP#

Formatted: Bullets and Numbering

There is one of these interfaces at front of each of the SP (buffer to stage pixel interpolators). This interface transmits the I,J data for pixel interpolation. For the entire system, two quads per clock are transferred to the 4 SPs, so each of these 4 interfaces transmits one half of a quad per clock. The interface below describes a half of a quad worth of data.

The actual data which is transferred per quad is
Ref Pix I => S4.20 Floating Point I value
Ref Pix J => S4.20 Floating Point J value
Delta Pix I (x3) => S4.8 Floating Point Delta I value
Delta Pix J (x3) => S4.8 Floating Point Delta J value
This equates to a total of 128 bits which transferred over 2 clocks
and therefor needs an interface 64 bits wide

Additionally, X,Y data (12-bit unsigned fixed) is conditionally sent across this data bus over the same wires in an additional clock. The X,Y data is sent on the lower 24 bits of the data bus with faceness in the msb.
Transfers across these interfaces are synchronized with the SC_SQ IJ Control Bus transfers.

The data transfer across each of these busses is controlled by a IJ_BUF_INUSE_COUNT in the SC. Each time the SC has sent a pixel vector's worth of data to the SPs, he will increment the IJ_BUF_INUSE_COUNT count. Prior to sending the next pixel vectors data, he will check to make sure the count is less than MAX_BUFER_MINUS_2, if not the SC will stall until the SQ returns a pipelined pulse to decrement the count when he has scheduled a buffer free. Note: We could/may optimize for the case of only sending only IJ to use all the buffers to pre-load more. Currently it is planned for the SP to hold 2 double buffers of I,J data and two buffers of X,Y data, so if either X,Y or Centers and Centroids are on, then the SC can send two Buffers.

In at least the initial version, the SC shall send 16 quads per pixel vector even if the vector is not full. This will increment buffer write address pointers correctly all the time. (We may revisit this for both the SX,SP,SQ and add a EndOfVector signal on all interfaces to quit early. We opted for the simple mode first with a belief that only the end of packet and multiple new vector signals should cause a partial vector and that this would not really be significant performance hit.)

Name	**Bits**	**Description**
SC_SP#_data	64	IJ information sent over 2 clocks (or X,Y in 24 LSBs with faceness in upper bit) **Type 0 or 1**, First clock I, second clk J Field ULC URC LLC LRC Bits [63:39] [38:26] [25:13] [12:0] Format SE4M20 SE4M8 SE4M8 SE4M8 **Type 2** Field Face X Y Bits [63] [23:12] [11:0] Format Bit Unsigned Unsigned
SC_SP#_valid	1	Valid
SC_SP#_last_quad	1	This bit will be set on the last transfer of data per quad.
SC_SP#_type	2	0 -> Indicates centroids 1 -> Indicates centers 2 -> Indicates X,Y Data and faceness on data bus The SC shall look at state data to determine how many types to send for the interpolation process.

The # is included for clarity in the spec and will be replaced with a prefix of u#_ in the verilog module statement for the SC and the SP block will have neither because the instantiation will insert the prefix.

27.2.2 SC_SQ

Formatted: Bullets and Numbering

This is the control information sent to the sequencer in order to synchronize and control the interpolation and/or loading data into the GPRs needed to execute a shader program on the sent pixels. This data will be sent over two clocks per transfer with 1 to 16 transfers. Therefore the bus (approx 92 bits) could be folded in half to approx 46 bits.

Name	**Bits**	**Description**
SC_SQ_data	46	Control Data sent to the SQ 1 clk transfers Event – valid data consist of event_id and state_id. Instruct SQ to post an event vector to send state id and event_id through request fifo and onto the reservation stations making sure state id and/or event_id gets back to the CP. Events only follow end of packets so no pixel vectors will be in progress. Empty Quad Mask – Transfer Control data consisting of pc_dealloc or new_vector. Receipt of this is to transfer pc_dealloc or new_vector without any valid quad data. New vector will always be posted to request fifo and pc_dealloc will be attached to any pixel vector outstanding or posted in request fifo if no valid quad outstanding. 2 clk transfers Quad Data Valid – Sending quad data with or without new_vector or pc_dealloc. New vector will be posted to request fifo with or without a pixel vector and pc_dealloc will be posted with a pixel vector unless none is in progress. In this case the pc_dealloc will be posted in the request queue. Filler quads will be transferred with The Quad mask set but the pixel corresponding pixel mask set to zero.
SC_SQ_valid	1	SC sending valid data, 2nd clk could be all zeroes

SC_SQ_data – first clock and second clock transfers are shown in the table below.

Name	**BitField**	**Bits**	**Description**
1st Clock Transfer			
SC_SQ_event	0	1	This transfer is a 1 clock event vector Force quad_mask = new_vector=pc_dealloc=0
SC_SQ_event_id	[2:1]	2	This field identifies the event 0 => denotes an End Of State Event 1 => TBD
SC_SQ_pc_dealloc	3	1	Deallocation token for the Parameter Cache
SC_SQ_new_vector	4	1	The SQ must wait for Vertex shader done count > 0 and after dispatching the Pixel Vector the SQ will decrement the count.
SC_SQ_quad_mask	[8:5]	4	Quad Write mask left to right SP0 => SP3
SC_SQ_end_of_prim	9	1	End Of the primitive
SC_SQ_state_id	[12:10]	3	State/constant pointer (6*3+3)
SC_SQ_pix_mask	[28:13]	16	Valid bits for all pixels SP0=>SP3 (UL,UR,LL,LR)
SC_SQ_prim_type	[31:29]	3	Stippled line and Real time command need to load tex cords from

			alternate buffer 000: Normal 100: Realtime 101: Line AA 110: Point AA (Sprite)
SC_SQ_pc_ptr0	[42:32]	11	Parameter Cache pointer for vertex 0
2nd Clock Transfer			
SC_SQ_pc_ptr1	[10:0]	11	Parameter Cache pointer for vertex 1
SC_SQ_pc_ptr2	[21:11]	11	Parameter Cache pointer for vertex 2
SC_SQ_lod_correct	[45:22]	24	LOD correction per quad (6 bits per quad)

Name	**Bits**	**Description**
SQ_SC_free_buff	1	Pipelined bit that instructs SC to decrement count of buffers in use.
SQ_SC_dec_cntr_cnt	1	Pipelined bit that instructs SC to decrement count of new vector and/or event sent to prevent SC from overflowing SQ interpolator/Reservation request fifo.

The scan converter will submit a partial vector whenever:

1.) He gets a primitive marked with an end of packet signal.

2.) A current pixel vector is being assembled with at least one or more valid quads and the vector has been marked for deallocate when a primitive marked new_vector arrives. The Scan Converter will submit a partial vector (up to 16quads with zero pixel mask to fill out the vector) prior to submitting the new_vector marker\primitive.

(This will prevent a hang which can be demonstrated when all primitives in a packet three vectors are culled except for a one quad primitive that gets marked pc_dealloc (vertices maximum size). In this case two new_vectors are submitted and processed, but then one valid quad with the pc_dealloc creates a vector and then the new would wait for another vertex vector to be processed, but the one being waited for could never export until the pc_dealloc signal made it through and thus the hang.)

Formatted: Bullets and Numbering

~~26.1.1 SC to SQ : IJ Control bus~~

Formatted: Bullets and Numbering

~~This is the control information sent to the sequencer in order to control the IJ fifos and all other information needed to execute a shader program on the sent pixels. This information is sent over 2 clocks, if SENDXY is asserted the next control packet is going to be ignored and XY information is going to be sent on the IJ bus (for the quads that where just sent). All pixels from the group of quads are from the same primitive, all quads of a vector are from the same render state.~~

~~26.1.2 SQ to SP: Interpolator bus~~

Formatted: Bullets and Numbering

~~26.1.3~~27.2.3 SQ to SX: Interpolator bus

Name	**Direction**	**Bits**	**Description**
SQ_~~SPx~~SXx_interp_flat_vtx	SQ→SPx	2	Provoking vertex for flat shading
SQ_S~~P~~Xx_interp_flat_gouraud	SQ→SPx	1	Flat or gouraud shading
SQ_S~~P~~Xx_interp_cyl_wrap	SQ→SPx	4	Wich channel needs to be cylindrical wrapped
SQ_S~~X~~Xx_ptr1~~mux0~~	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_ptr2~~mux1~~	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_ptr3~~mux2~~	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_~~RT_switch~~rt_sel	SQ→SXx	1	Selects between RT and Normal data
SQ_SXx_pc_wr_en	SQ→SXx	1	Write enable for the PC memories
SQ_SXx_pc_wr_addr	SQ→SXx	7	Write address for the PCs
SQ_SXx_pc_cmask	SQ→SXx	4	Channel mask

~~26.1.4~~27.2.4 SQ to SP: Staging Register Data

Formatted: Bullets and Numbering

This is a broadcast bus that sends the VSISR information to the staging registers of the shader pipes.

Name	**Direction**	**Bits**	**Description**

SQ_SPx_~~vgt_vsisr~~vsr_data	SQ→SPx	96	Pointers of indexes or HOS surface information
SQ_SPx_~~vgt_vsisr~~vsr_double	SQ→SPx	1	0: Normal 96 bits per vert 1: double 192 bits per vert
SQ_SP0__~~data_~~valid	SQ→SP0	1	Data is valid
SQ_SP1__~~data_~~valid	SQ→SP1	1	Data is valid
SQ_SP2__~~data_~~valid	SQ→SP2	1	Data is valid
SQ_SP3__~~data_~~valid	SQ→SP3	1	Data is valid

~~26.1.5~~27.2.5 *~~PA~~ VGT to SQ : Vertex interface*

Formatted: Bullets and Numbering

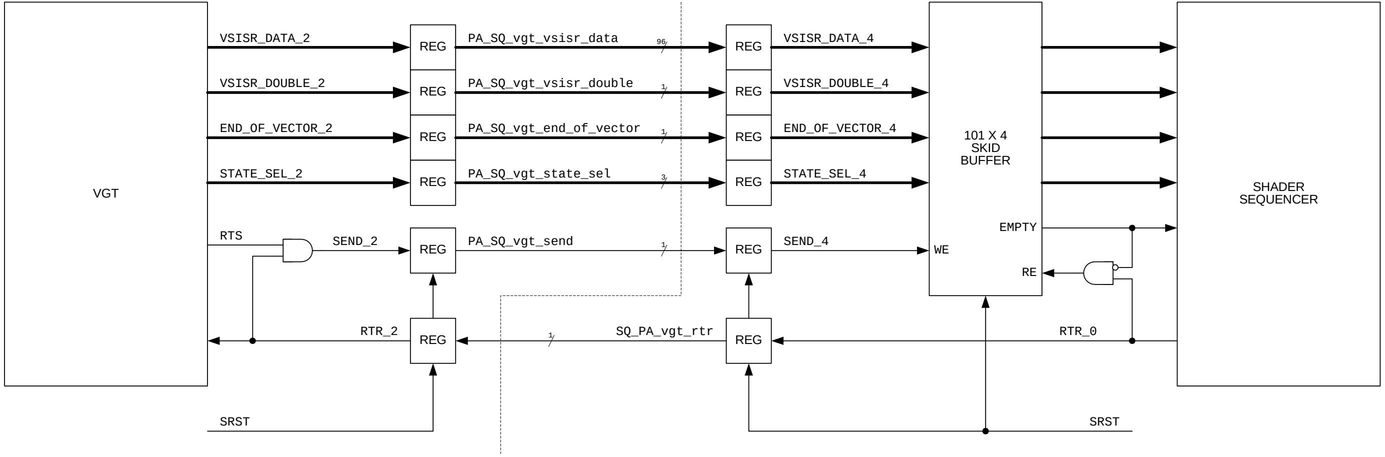
~~26.1.5.1~~27.2.5.1 Interface Signal Table

The area difference between the two methods is not sufficient to warrant complicating the interface or the state requirements of the VSISRs. **Therefore, the POR for this interface is that the VGT will transmit the data to the VSISRs (via the Shader Sequencer) in full, 32-bit floating-point format.** The VGT can transmit up to six 32-bit floating-point values to each VSISR where four or more values require two transmission clocks. The data bus is 96 bits wide.

Name	Bits	Description
~~PA~~VGT_SQ__~~vgt_~~vsisr_data	96	Pointers of indexes or HOS surface information
VGT~~PA~~_SQ__~~vgt_~~vsisr_double	1	0: Normal 96 bits per vert 1: double 192 bits per vert
VGT~~PA_~~_SQ__~~vgt_~~end_of_vector	1	Indicates the last VSISR data set for the current process vector (for double vector data, "end_of_vector" is set on the second vector)
VGT~~PA~~_SQ__~~vgt_vsisr~~indx_valid	1	Vsisr data is valid
VGT~~PA~~_SQ__~~vgt_~~state	3	Render State (6*3+3 for constants). This signal is guaranteed to be correct when "~~PA~~VGT_SQ_vgt_end_of_vector" is high.
VGT~~PA~~_SQ__~~vgt_~~send	1	Data on the VGT_SQ is valid receive (see write-up for standard R400 SEND/RTR interface handshaking)
SQ_VGT_~~PA_vgt_~~rtr	1	Ready to receive (see write-up for standard R400 SEND/RTR interface handshaking)

~~26.1.5.2~~27.2.5.2 Interface Diagrams

Formatted: Bullets and Numbering



VGT
VSISR_DATA_2
VSISR_DOUBLE_2
END_OF_VECTOR_2
STATE_SEL_2
RTS
SEND_2
RTR_2
SRST
REG
PA_SQ_vgt_vsisr_data
96
PA_SQ_vgt_vsisr_double
1
PA_SQ_vgt_end_of_vector
1
PA_SQ_vgt_state_sel
3
PA_SQ_vgt_send
1
SQ_PA_vgt_rtr
1
VSISR_DATA_4
VSISR_DOUBLE_4
END_OF_VECTOR_4
STATE_SEL_4
SEND_4
101 X 4
SKID
BUFFER
EMPTY
WE
RE
RTR_0
SRST
SHADER
SEQUENCER

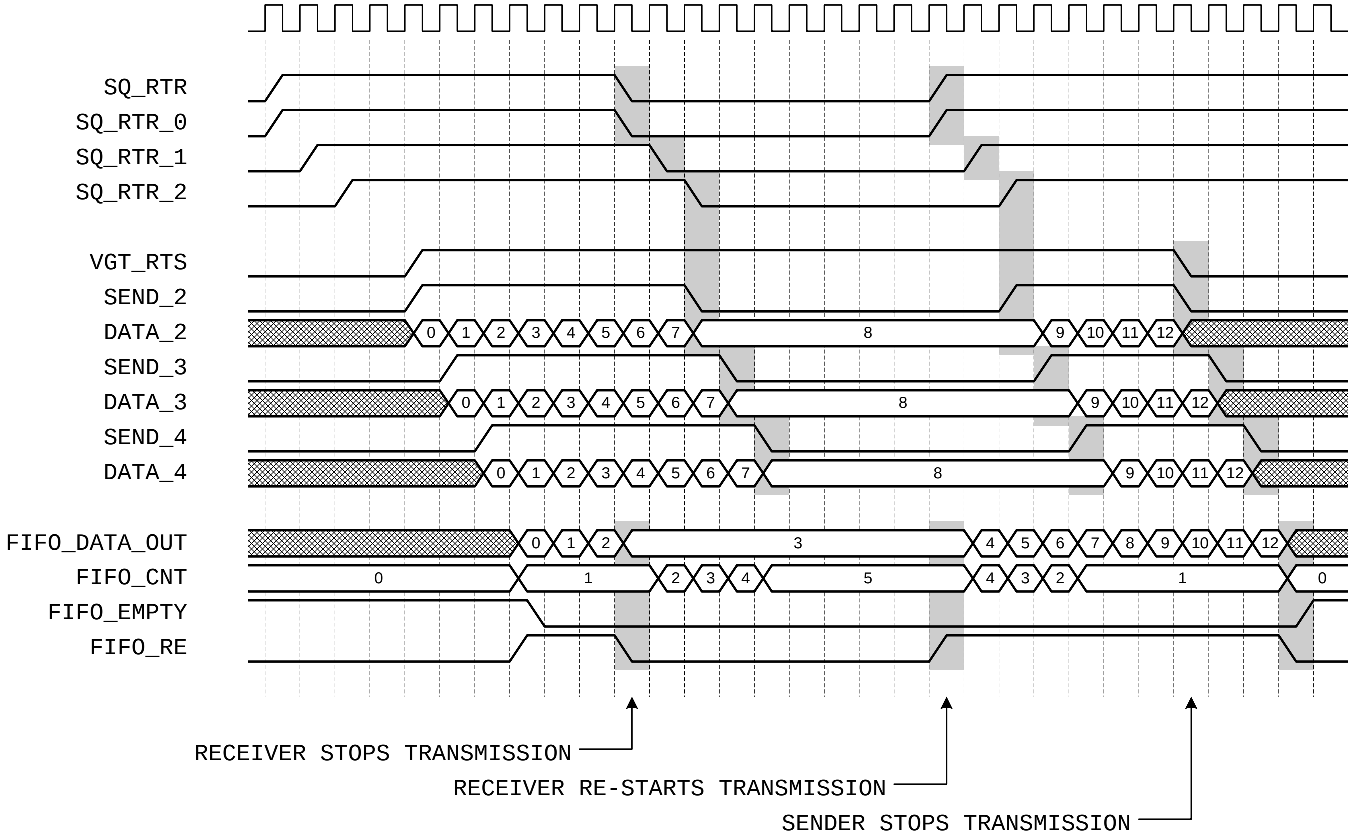


Figure 1. Detailed Logical Diagram for PA_SQ_vgt Interface.

~~26.1.6~~27.2.6 SQ to CP: State report

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_CP_vrtx_ state	SEQ→CP	3	Oldest vertex state still in the pipe
SQ_CP_pix_state	SEQ→CP	3	Oldest pixel state still in the pipe

~~26.1.7~~27.2.7 SQ to SX: Control bus

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SXx_exp_~~Pixel~~pix	SQ→SXx	1	1: Pixel 0: Vertex
SQ_SXx_exp_c~~C~~lause	SQ→SXx	3	Clause number, which is needed for vertex clauses
SQ_SXx_exp_s~~S~~tate	SQ→SXx	3	State ID
SQ_SXx_exp_~~exportID~~alu_id	SQ→SXx	1	ALU ID

These fields are sent ~~synchronously with SP export data, described in SP0 →SX0 interface~~every time the sequencer picks an exporting clause for execution.

~~26.1.8~~27.2.8 SX to SQ : Output file control

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SXx_SQ_~~Export~~exp_count_rdy	SXx→SQ	1	Raised by SX0 to indicate that the following two fields reflect the result of the most recent export
SXx_SQ_~~Export~~exp_~~P~~position_space	SXx→SQ	1	Specifies whether there is room for another position.
SXx_SQ_exp~~Export~~_b~~B~~uffer_space	SXx→SQ	7	Specifies the space available in the output buffers. 0: buffers are full 1: 2K-bits available (32-bits for each of the 64 pixels in a clause) ... 64: 128K-bits available (16 128-bit entries for each of 64 pixels) 65-127: RESERVED

~~26.1.9~~27.2.9 SQ to TP: Control bus

Formatted: Bullets and Numbering

Once every clock, the fetch unit sends to the sequencer on which clause it is now working and if the data in the GPRs is ready or not. This way the sequencer can update the fetch counters for the reservation station fifos. The sequencer also provides the instruction and constants for the fetch to execute and the address in the register file where to write the fetch return data.

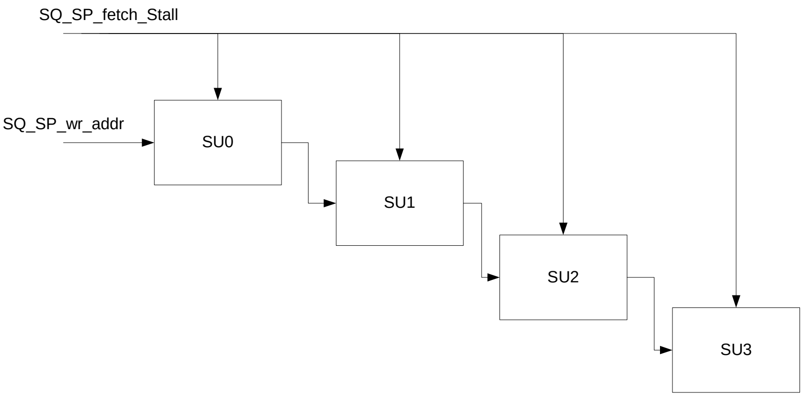
Name	Direction	Bits	Description
TPx_SQ_data_rdy	TPx→ SQ	1	Data ready
TPx_SQ_clause_num	TPx→ SQ	3	Clause number
TPx_SQ_t~~T~~ype	TPx→ SQ	1	Type of data sent (0:PIXEL, 1:VERTEX)
SQ_TPx_send	SQ→TPx	1	Sending valid data
SQ_TPx_const	SQ→TPx	48	Fetch state sent over 4 clocks (192 bits total)
SQ_TPx_~~instuct~~instr	SQ→TPx	24	Fetch instruction sent over 4 clocks
SQ_TPx_end_of_clause	SQ→TPx	1	Last instruction of the clause
SQ_TPx_Type	SQ→TPx	1	Type of data sent (0:PIXEL, 1:VERTEX)
SQ_TPx_phase	SQ→TPx	2	Write phase signal
SQ_TP0_lod_correct	SQ→TP0	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP0_pmask	SQ→TP0	4	Pixel mask 1 bit per pixel
SQ_TP1_lod_correct	SQ→TP1	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP1_pmask	SQ→TP1	4	Pixel mask 1 bit per pixel
SQ_TP2_lod_correct	SQ→TP2	6	LOD correct 3 bits per comp 2 components per quad

SQ_TP2_pmask	SQ→TP2	4	Pixel mask 1 bit per pixel
SQ_TP3_lod_correct	SQ→TP3	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP3_pmask	SQ→TP3	4	Pixel mask 1 bit per pixel
SQ_TPx_clause_num	SQ→TPx	3	Clause number
SQ_TPx_write_gpr_index	SQ->TPx	7	Index into Register file for write of returned Fetch Data

~~26.1.10~~27.2.10 *TP to SQ: Texture stall*

Formatted: Bullets and Numbering

The TP sends this signal to the SQ when its input buffer is full. The SQ is going to send it to the SP X clocks after reception (maximum of 3 clocks of pipeline delay).



Name	Direction	Bits	Description
TP_SQ_fetch_stall	TP→ SQ	1	Do not send more texture request if asserted

~~26.1.11~~27.2.11 *SQ to SP: Texture stall*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SPx_fetch_stall	SQ→SPx	1	Do not send more texture request if asserted

~~26.1.12~~27.2.12 *SQ to SP: GPR~~, Parameter cache control~~ and auto counter*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SPx_gpr_wr_addr	SQ→SPx	7	Write address
SQ_SPx_gpr_rd_addr	SQ→SPx	7	Read address
SQ_SPx_gpr_r~~e~~d_~~addr~~en	SQ→SPx	1	Read Enable
SQ_SPx_gpr_~~we~~wr_~~addr~~en	SQ→SPx	1	Write Enable for the GPRs
SQ_SPx_gpr_phase_mux	SQ→SPx	2	The phase mux (arbitrates between inputs, ALU SRC reads and writes)
SQ_SPx_channel_mask	SQ→SPx	4	The channel mask
SQ_SP0_pixel_mask	SQ→SP0	4	The pixel mask
SQ_SP1_pixel_mask	SQ→SP1	4	The pixel mask
SQ_SP2_pixel_mask	SQ→SP2	4	The pixel mask
SQ_SP3_pixel_mask	SQ→SP3	4	The pixel mask
SQ_SPx_gpr_input_mux	SQ→SPx	2	When the phase mux selects the inputs this tells from which source to read from: Interpolated data, VTX0, VTX1, autogen counter.
SQ_SPx_~~index~~auto_count	SQ→SPx	12?	~~Index~~ Auto count generated by the SQ, common for all shader pipes

~~26.1.13~~27.2.13 SQ to SPx: Instructions

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SPx_instr~~uct~~_start	SQ→SPx	1	Instruction start
SQ_SP_instr~~uct~~	SQ→SPx	21	Transferred over 4 cycles 0: SRC A Select 2:0 SRC A Argument Modifier 3:3 SRC A swizzle 11:4 VectorDst 17:12 Unused 20:18 -- - 1: SRC B Select 2:0 SRC B Argument Modifier 3:3 SRC B swizzle 11:4 ScalarDst 17:12 Unused 20:18 -- - 2: SRC C Select 2:0 SRC C Argument Modifier 3:3 SRC C swizzle 11:4 Unused 20:12 -- - 3: Vector Opcode 4:0 Scalar Opcode 10:5 Vector Clamp 11:11 Scalar Clamp 12:12 Vector Write Mask 16:13 Scalar Write Mask 20:17
SQ_SPx_exp_alu_id	SQ→SPx	1	ALU ID
SQ_SPx_exporting	SQ→SPx	2	0: Not Exporting 1: Vector Exporting 2: Scalar Exporting
SQ_SPx_stall	SQ→SPx	1	Stall signal
SQ_SP0_exp~~ort~~_pvalid	SQ→SP0	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP1_ exp~~ort~~_pvalid	SQ→SP1	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP2_ exp~~ort~~_pvalid	SQ→SP2	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP3_ exp~~ort~~_pvalid	SQ→SP3	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock

~~26.1.14~~27.2.14 SP to SQ: Constant address load/ Predicate Set

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SP0_SQ_const_addr	SP0→SQ	36	Constant address load / predicate vector load (4 bits only) to the sequencer
SP0_SQ_valid	SP0→SQ	1	Data valid
SP1_SQ_const_addr	SP1→SQ	36	Constant address load / predicate vector load (4 bits only) to the sequencer

SP1_SQ_valid	SP1→SQ	1	Data valid
SP2_SQ_const_addr	SP2→SQ	36	Constant address load / predicate vector load (4 bits only) to the sequencer
SP2_SQ_valid	SP2→SQ	1	Data valid
SP3_SQ_const_addr	SP3→SQ	36	Constant address load / predicate vector load (4 bits only) to the sequencer
SP3_SQ_valid	SP3→SQ	1	Data valid

~~26.1.15~~27.2.15 *SQ to SPx: constant broadcast*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SPx_constt~~tant~~	SQ→SPx	128	Constant broadcast

~~26.1.16~~27.2.16 *SP0 to SQ: Kill vector load*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SP0_SQ_kill_vect	SP0→SQ	4	Kill vector load
SP1_SQ_kill_vect	SP1→SQ	4	Kill vector load
SP2_SQ_kill_vect	SP2→SQ	4	Kill vector load
SP3_SQ_kill_vect	SP3→SQ	4	Kill vector load

~~26.1.17~~27.2.17 *SQ to CP: RBBM bus*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_RBB_rs	SQ→CP	1	Read Strobe
SQ_RBB_rd	SQ→CP	32	Read Data
SQ_RBBM_nrtrtr	SQ→CP	1	Optional
SQ_RBBM_rtr	SQ→CP	1	Real-Time (Optional)

~~26.1.18~~27.2.18 *CP to SQ: RBBM bus*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
rbbm_we	CP→SQ	1	Write Enable
rbbm_a	CP→SQ	15	Address -- Upper Extent is TBD (16:2)
rbbm_wd	CP→SQ	32	Data
rbbm_be	CP→SQ	4	Byte Enables
rbbm_re	CP→SQ	1	Read Enable
rbb_rs0	CP→SQ	1	Read Return Strobe 0
rbb_rs1	CP→SQ	1	Read Return Strobe 1
rbb_rd0	CP→SQ	32	Read Data 0
rbb_rd1	CP→SQ	32	Read Data 0
RBBM_SQ_soft_reset	CP→SQ	1	Soft Reset

~~27.~~28. Examples of program executions

Formatted: Bullets and Numbering

~~27.1.1~~28.1.1 *Sequencer Control of a Vector of Vertices*

1. PA sends a vector of 64 vertices (actually vertex indices – 32 bits/index for 2048 bit total) to the RE's Vertex FIFO
   - state pointer as well as tag into position cache is sent along with vertices
   - space was allocated in the position cache for transformed position before the vector was sent
   - **also before the vector is sent to the RE, the CP has loaded the global instruction store with the vertex shader program (using the MH?)**
   - The vertex program is assumed to be loaded when we receive the vertex vector.
     - the SEQ then accesses the IS base for this shader using the local state pointer (provided to all sequencers by the RBBM when the CP is done loading the program)
2. SEQ arbitrates between the Pixel FIFO and the Vertex FIFO – basically the Vertex FIFO always has priority
   - at this point the vector is removed from the Vertex FIFO

- the arbiter is not going to select a vector to be transformed if the parameter cache is full unless the pipe as nothing else to do (ie no pixels are in the pixel fifo).

3. SEQ allocates space in the SP register file for index data plus GPRs used by the program
   - the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer that came down with the vertices
   - SEQ will not send vertex data until space in the register file has been allocated

4. SEQ sends the vector to the SP register file over the RE_SP interface (which has a bandwidth of 2048 bits/cycle)
   - the 64 vertex indices are sent to the 64 register files over 4 cycles
     - RF0 of SU0, SU1, SU2, and SU3 is written the first cycle
     - RF1 of SU0, SU1, SU2, and SU3 is written the second cycle
     - RF2 of SU0, SU1, SU2, and SU3 is written the third cycle
     - RF3 of SU0, SU1, SU2, and SU3 is written the fourth cycle
   - the index is written to the least significant 32 bits **(floating point format?) (what about compound indices)** of the 128-bit location within the register file (w); the remaining data bits are set to zero (x, y, z)

5. SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of fetch state machine 0, or TSM0 FIFO)
   - the control packet contains the state pointer, the tag to the position cache and a register file base pointer.

6. TSM0 accepts the control packet and fetches the instructions for fetch clause 0 from the global instruction store
   - TSM0 was first selected by the TSM arbiter before it could start

7. all instructions of fetch clause 0 are issued by TSM0

8. the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)
   - TSM0 does not wait for requests made to the Fetch Unit to complete; it passes the register file write index for the fetch data to the TU, which will write the data to the RF as it is received
   - once the TU has written all the data to the register files, it increments a counter that is associated with ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go ahead start to execute the ALU clause

9. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store

10. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of fetch state machine 1, or TSM1 FIFO)

11. the control packet continues to travel down the path of reservation stations until all clauses have been executed
    - position can be exported in ALU clause 3 (or 4?); the data (and the tag) is sent over a position bus (which is shared with all four shader pipes) back to the PA's position cache
    - A parameter cache pointer is also sent along with the position data. This tells to the PA where the data is going to be in the parameter cache.
      - there is a position export FIFO in the SP that buffers position data before it gets sent back to the PA
      - the ASM arbiter will prevent a packet from starting an exporting clause if the position export FIFO is full
    - parameter data is exported in clause 7 (as well as position data if it was not exported earlier)
      - parameter data is sent to the Parameter Cache over a dedicated bus
      - the SEQ allocates storage in the Parameter Cache, and the SEQ deallocates that space when there is no longer a need for the parameters (it is told by the PA when using a token).
      - the ASM arbiter will prevent a packet from starting on ASM7 if the parameter cache (or the position buffer if position is being exported) is full

12. after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program

~~27.1.2~~28.1.2 *Sequencer Control of a Vector of Pixels*

Formatted: Bullets and Numbering

1. **As with vertex shader programs, pixel shaders are loaded into the global instruction store by the CP**
   - At this point it is assumed that the pixel program is loaded into the instruction store and thus ready to be read.

2. the RE’s Pixel FIFO is loaded with the barycentric coordinates for pixel quads by the detailed walker
   - the state pointer and the LOD correction bits are also placed in the Pixel FIF0
   - the Pixel FIFO is wide enough to source four quad’s worth of barycentrics per cycle
3. SEQ arbitrates between Pixel FIFO and Vertex FIFO – when there are no vertices pending OR there is no space left in the register files for vertices, the Pixel FIFO is selected
4. SEQ allocates space in the SP register file for all the GPRs used by the program
   - the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer
   - SEQ will not allow interpolated data to be sent to the shader until space in the register file has been allocated
5. SEQ controls the transfer of interpolated data to the SP register file over the RE_SP interface (which has a bandwidth of 2048 bits/cycle). See interpolated data bus diagrams for details.
6. SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of fetch state machine 0, or TSM0 FIFO)
   - note that there is a separate set of reservation stations/arbiters/state machines for vertices and for pixels
   - the control packet contains the state pointer, the register file base pointer, and the LOD correction bits
   - all other information (such as quad address for example) travels in a separate FIFO
7. TSM0 accepts the control packet and fetches the instructions for fetch clause 0 from the global instruction store
   - TSM0 was first selected by the TSM arbiter before it could start
8. all instructions of fetch clause 0 are issued by TSM0
9. the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)
   - TSM0 does not wait for fetch requests made to the Fetch Unit to complete; it passes the register file write index for the fetch data to the TU, which will write the data to the RF as it is received
   - once the TU has written all the data for a particular clause to the register files, it increments a counter that is associated with the ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go ahead and pop the FIFO and start to execute the ALU clause
10. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store
11. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of fetch state machine 1, or TSM1 FIFO)
12. the control packet continues to travel down the path of reservation stations until all clauses have been executed
    - pixel data is exported in the last ALU clause (clause 7)
      - it is sent to an output FIFO where it will be picked up by the render backend
      - the ASM arbiter will prevent a packet from starting on ASM7 if the output FIFO is full
13. after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program

~~27.1.3~~28.1.3 *Notes*

Formatted: Bullets and Numbering

14. The state machines and arbiters will operate ahead of time so that they will be able to immediately start the real threads or stall.
15. The register file base pointer for a vector needs to travel with the vector through the reservation stations, but the instruction store base pointer does not – this is because the RF pointer is different for all threads, but the IS pointer is only different for each state and thus can be accessed via the state pointer.

~~28.~~29. Open issues

Formatted: Bullets and Numbering

Need to do some testing on the size of the register file as well as on the register file allocation method (dynamic VS static).

Saving power?

Parameter caches in SX?

Using both IJ buffers for center + centroid interpolation?

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 2015~~19~~ April, 200225 March	GEN-CXXXXX-REVA	1 of 52

Author: Laurent Lefebvre

Issue To:	**Copy No:**

R400 Sequencer Specification

SQ

Version 1.11~~0~~

Overview: This is an architectural specification for the R400 Sequencer block (SEQ). It provides an overview of the required capabilities and expected uses of the block. It also describes the block interfaces, internal sub-blocks, and provides internal state diagrams.

AUTOMATICALLY UPDATED FIELDS:
Document Location: C:\perforce\r400\doc_lib\design\blocks\sq\R400_Sequencer.doc
Current Intranet Search Title: R400 Sequencer Specification

APPROVALS	
Name/Dept	Signature/Date

Remarks:

THIS DOCUMENT CONTAINS CONFIDENTIAL INFORMATION THAT COULD BE SUBSTANTIALLY DETRIMENTAL TO THE INTEREST OF ATI TECHNOLOGIES INC. THROUGH UNAUTHORIZED USE OR DISCLOSURE.

ATI 2027
LG v. ATI
IPR2015-00326

Table Of Contents

1. OVERVIEW .... **8~~6~~**
1.1 Top Level Block Diagram .... 10~~8~~
1.2 Data Flow graph (SP) .... 12~~10~~
1.3 Control Graph .... 13~~11~~
2. INTERPOLATED DATA BUS .... **13~~11~~**
3. INSTRUCTION STORE .... **16~~14~~**
4. SEQUENCER INSTRUCTIONS .... **18~~14~~**
5. CONSTANT STORES .... **18~~14~~**
5.1 Memory organizations .... 18~~14~~
5.2 Management of the Control Flow Constants .... 18~~15~~
5.3 Management of the re-mapping tables .... 18~~15~~
5.3.1 R400 Constant management .... 18~~15~~
5.3.2 Proposal for R400LE constant management .... 19~~15~~
5.3.3 Dirty bits .... 21~~17~~
5.3.4 Free List Block .... 21~~17~~
5.3.5 De-allocate Block .... 22~~18~~
5.3.6 Operation of Incremental model .... 22~~18~~
5.4 Constant Store Indexing .... 22~~18~~
5.5 Real Time Commands .... 23~~19~~
5.6 Constant Waterfalling .... 23~~19~~
6. LOOPING AND BRANCHES .... **24~~20~~**
6.1 The controlling state. .... 24~~20~~
6.2 The Control Flow Program .... 24~~20~~
6.3 Data dependant predicate instructions .... 26~~22~~
6.4 HW Detection of PV,PS .... 27~~23~~
6.5 Register file indexing .... 27~~23~~
6.6 Predicated Instruction support for Texture clauses .... 27~~23~~
6.7 Debugging the Shaders .... 27~~23~~
6.7.1 Method 1: Debugging registers .... 28~~23~~
6.7.2 Method 2: Exporting the values in the GPRs (12) .... 28~~24~~
7. PIXEL KILL MASK .... **28~~24~~**
8. MULTIPASS VERTEX SHADERS (HOS) .... **29~~24~~**
9. REGISTER FILE ALLOCATION .... **29~~24~~**
10. FETCH ARBITRATION .... **30~~26~~**
11. ALU ARBITRATION .... **30~~26~~**
12. HANDLING STALLS .... **31~~27~~**
13. CONTENT OF THE RESERVATION STATION FIFOS .... **31~~27~~**
14. THE OUTPUT FILE .... **31~~27~~**
15. IJ FORMAT .... **31~~27~~**
15.1 Interpolation of constant attributes .... 32~~28~~
16. STAGING REGISTERS .... **32~~28~~**
17. THE PARAMETER CACHE .... **34~~30~~**
18. VERTEX POSITION EXPORTING .... **35~~30~~**

19. EXPORTING ARBITRATION ..... 35~~30~~
20. EXPORTING RULES ..... 35~~30~~
20.1 Parameter caches exports ..... 35~~30~~
20.2 Memory exports ..... 35~~30~~
20.3 Position exports ..... 35~~30~~
21. EXPORT TYPES ..... 35~~30~~
21.1 Vertex Shading ..... 35~~31~~
21.2 Pixel Shading ..... 35~~31~~
22. SPECIAL INTERPOLATION MODES ..... 36~~31~~
22.1 Real time commands ..... 36~~31~~
22.2 Sprites/ XY screen coordinates/ FB information ..... 36~~31~~
22.3 Auto generated counters ..... 36~~32~~
22.3.1 Vertex shaders ..... 37~~32~~
22.3.2 Pixel shaders ..... 37~~32~~
23. STATE MANAGEMENT ..... 37~~33~~
23.1 Parameter cache synchronization ..... 37~~33~~
24. XY ADDRESS IMPORTS ..... 37~~33~~
24.1 Vertex indexes imports ..... 38~~33~~
25. REGISTERS ..... 38~~33~~
25.1 Control ..... 38~~33~~
25.2 Context ..... 38~~33~~
26. DEBUG REGISTERS ..... 39~~34~~
26.1 Context ..... 39~~34~~
26.2 Control ..... 39~~34~~
27. INTERFACES ..... 39~~35~~
27.1 External Interfaces ..... 39~~35~~
27.2 SC to SP Interfaces ..... 39~~35~~
27.2.1 SC_SP# ..... 39~~35~~
27.2.2 SC_SQ ..... 40~~36~~
27.2.3 SQ to SX: Interpolator bus ..... 42~~37~~
27.2.4 SQ to SP: Staging Register Data ..... 42~~37~~
27.2.5 VGT to SQ : Vertex interface ..... 42~~38~~
27.2.6 SQ to SX: Control bus ..... 46~~41~~
27.2.7 SX to SQ : Output file control ..... 46~~41~~
27.2.8 SQ to TP: Control bus ..... 46~~41~~
27.2.9 TP to SQ: Texture stall ..... 47~~42~~
27.2.10 SQ to SP: Texture stall ..... 47~~42~~
27.2.11 SQ to SP: GPR and auto counter ..... 47~~42~~
27.2.12 SQ to SPx: Instructions ..... 48~~43~~
27.2.13 SP to SQ: Constant address load/ Predicate Set ..... 48~~43~~
27.2.14 SQ to SPx: constant broadcast ..... 49~~44~~
27.2.15 SP0 to SQ: Kill vector load ..... 49~~44~~
27.2.16 SQ to CP: RBBM bus ..... 49~~44~~

27.2.17 CP to SQ: RBBM bus .... 49~~44~~
27.2.18 SQ to CP: State report .... 49~~44~~
28. OPEN ISSUES .... 52~~44~~
~~**1. OVERVIEW .... 6**~~
~~1.1 Top Level Block Diagram .... 8~~
~~1.2 Data Flow graph (SP) .... 10~~
~~1.3 Control Graph .... 11~~
~~**2. INTERPOLATED DATA BUS .... 11**~~
~~**3. INSTRUCTION STORE .... 14**~~
~~**4. SEQUENCER INSTRUCTIONS .... 16**~~
~~**5. CONSTANT STORES .... 16**~~
~~5.1 Memory organizations .... 16~~
~~5.2 Management of the Control Flow Constants .... 16~~
~~5.3 Management of the re-mapping tables .... 16~~
~~5.3.1 R400 Constant management .... 16~~
~~5.3.2 Proposal for R400LE constant management .... 17~~
~~5.3.3 Dirty bits .... 19~~
~~5.3.4 Free List Block .... 19~~
~~5.3.5 De-allocate Block .... 20~~
~~5.3.6 Operation of Incremental model .... 20~~
~~5.4 Constant Store Indexing .... 20~~
~~5.5 Real Time Commands .... 21~~
~~5.6 Constant Waterfalling .... 21~~
~~**6. LOOPING AND BRANCHES .... 22**~~
~~6.1 The controlling state. .... 22~~
~~6.2 The Control Flow Program .... 22~~
~~6.3 Data dependant predicate instructions .... 24~~
~~6.4 HW Detection of PV,PS .... 25~~
~~6.5 Register file indexing .... 25~~
~~6.6 Predicated Instruction support for Texture clauses .... 25~~
~~6.7 Debugging the Shaders .... 25~~
~~6.7.1 Method 1: Debugging registers .... 25~~
~~6.7.2 Method 2: Exporting the values in the GPRs (12) .... 26~~
~~**7. PIXEL KILL MASK .... 26**~~
~~**8. MULTIPASS VERTEX SHADERS (HOS) .... 26**~~
~~**9. REGISTER FILE ALLOCATION .... 26**~~
~~**10. FETCH ARBITRATION .... 28**~~
~~**11. ALU ARBITRATION .... 28**~~
~~**12. HANDLING STALLS .... 29**~~
~~**13. CONTENT OF THE RESERVATION STATION FIFOS .... 29**~~
~~**14. THE OUTPUT FILE .... 29**~~
~~**15. IJ FORMAT .... 29**~~
~~15.1 Interpolation of constant attributes .... 30~~
~~**16. STAGING REGISTERS .... 30**~~
~~**17. THE PARAMETER CACHE .... 32**~~

~~18. VERTEX POSITION EXPORTING........32~~
~~19. EXPORTING ARBITRATION........32~~
~~20. EXPORT TYPES........32~~
~~20.1 Vertex Shading........32~~
~~20.2 Pixel Shading........33~~
~~21. SPECIAL INTERPOLATION MODES........33~~
~~21.1 Real time commands........33~~
~~21.2 Sprites/ XY screen coordinates/ FB information........33~~
~~21.3 Auto generated counters........34~~
~~21.3.1 Vertex shaders........34~~
~~21.3.2 Pixel shaders........34~~
~~22. STATE MANAGEMENT........34~~
~~22.1 Parameter cache synchronization........34~~
~~23. XY ADDRESS IMPORTS........35~~
~~23.1 Vertex indexes imports........35~~
~~24. REGISTERS........35~~
~~24.1 Control........35~~
~~24.2 Context........35~~
~~25. DEBUG REGISTERS........36~~
~~25.1 Context........36~~
~~25.2 Control........36~~
~~26. INTERFACES........36~~
~~26.1 External Interfaces........36~~
~~26.1.1 SC to SQ : IJ Control bus........37~~
~~26.1.2 SQ to SP: Interpolator bus........37~~
~~26.1.3 SQ to SX: Interpolator bus........37~~
~~26.1.4 SQ to SP: Staging Register Data........38~~
~~26.1.5 PA to SQ : Vertex interface........38~~
~~26.1.6 SQ to CP: State report........41~~
~~26.1.7 SQ to SX: Control bus........41~~
~~26.1.8 SX to SQ : Output file control........41~~
~~26.1.9 SQ to TP: Control bus........41~~
~~26.1.10 TP to SQ: Texture stall........42~~
~~26.1.11 SQ to SP: Texture stall........42~~
~~26.1.12 SQ to SP: GPR and auto counter........42~~
~~26.1.13 SQ to SPx: Instructions........43~~
~~26.1.14 SP to SQ: Constant address load/ Predicate Set........44~~
~~26.1.15 SQ to SPx: constant broadcast........44~~
~~26.1.16 SP0 to SQ: Kill vector load........44~~
~~26.1.17 SQ to CP: RBBM bus........44~~
~~26.1.18 CP to SQ: RBBM bus........44~~
~~27. EXAMPLES OF PROGRAM EXECUTIONS........45~~

~~27.1.1 Sequencer Control of a Vector of Vertices .... 45~~
~~27.1.2 Sequencer Control of a Vector of Pixels .... 46~~
~~27.1.3 Notes .... 47~~
~~28. OPEN ISSUES .... 47~~

Revision Changes:

Rev 0.1 (Laurent Lefebvre) Date: May 7, 2001	First draft.
Rev 0.2 (Laurent Lefebvre) Date : July 9, 2001	Changed the interfaces to reflect the changes in the SP. Added some details in the arbitration section.
Rev 0.3 (Laurent Lefebvre) Date : August 6, 2001	Reviewed the Sequencer spec after the meeting on August 3, 2001.
Rev 0.4 (Laurent Lefebvre) Date : August 24, 2001	Added the dynamic allocation method for register file and an example (written in part by Vic) of the flow of pixels/vertices in the sequencer.
Rev 0.5 (Laurent Lefebvre) Date : September 7, 2001	Added timing diagrams (Vic)
Rev 0.6 (Laurent Lefebvre) Date : September 24, 2001	Changed the spec to reflect the new R400 architecture. Added interfaces.
Rev 0.7 (Laurent Lefebvre) Date : October 5, 2001	Added constant store management, instruction store management, control flow management and data dependant predication.
Rev 0.8 (Laurent Lefebvre) Date : October 8, 2001	Changed the control flow method to be more flexible. Also updated the external interfaces.
Rev 0.9 (Laurent Lefebvre) Date : October 17, 2001	Incorporated changes made in the 10/18/01 control flow meeting. Added a NOP instruction, removed the conditional_execute_or_jump. Added debug registers.
Rev 1.0 (Laurent Lefebvre) Date : October 19, 2001	Refined interfaces to RB. Added state registers.
Rev 1.1 (Laurent Lefebvre) Date : October 26, 2001	Added SEQ→SP0 interfaces. Changed delta precision. Changed VGT→SP0 interface. Debug Methods added.
Rev 1.2 (Laurent Lefebvre) Date : November 16, 2001	Interfaces greatly refined. Cleaned up the spec.
Rev 1.3 (Laurent Lefebvre) Date : November 26, 2001	Added the different interpolation modes.
Rev 1.4 (Laurent Lefebvre) Date : December 6, 2001	Added the auto incrementing counters. Changed the VGT→SQ interface. Added content on constant management. Updated GPRs.
Rev 1.5 (Laurent Lefebvre) Date : December 11, 2001	Removed from the spec all interfaces that weren't directly tied to the SQ. Added explanations on constant management. Added PA→SQ synchronization fields and explanation.
Rev 1.6 (Laurent Lefebvre) Date : January 7, 2002	Added more details on the staging register. Added detail about the parameter caches. Changed the call instruction to a Conditionnal_call instruction. Added details on constant management and updated the diagram.
Rev 1.7 (Laurent Lefebvre) Date : February 4, 2002	Added Real Time parameter control in the SX interface. Updated the control flow section.
Rev 1.8 (Laurent Lefebvre) Date : March 4, 2002	New interfaces to the SX block. Added the end of clause modifier, removed the end of clause instructions.
Rev 1.9 (Laurent Lefebvre) Date : March 18, 2002	Rearangement of the CF instruction bits in order to ensure byte alignement.
Rev 1.10 (Laurent Lefebvre) Date : March 25, 2002	Updated the interfaces and added a section on exporting rules.
Rev 1.11 (Laurent Lefebvre) Date :	Added CP state report interface. Last version of the spec with the old control flow scheme

1. Overview

The sequencer is based on the R300 design. It chooses two ALU clauses and a fetch clause to execute, and executes all of the instructions in a clause before looking for a new clause of the same type. Two ALU clauses are executed interleaved to hide the ALU latency. Each vector will have eight fetch and eight ALU clauses, but clauses do not need to contain instructions. A vector of pixels or vertices ping-pongs along the sequencer FIFO, bouncing from fetch reservation station to alu reservation station. A FIFO exists between each reservation stage, holding up vectors until the vector currently occupying a reservation station has left. A vector at a reservation station can be chosen to execute. The sequencer looks at all eight alu reservation stations to choose an alu clause to execute and all eight fetch stations to choose a fetch clause to execute. The arbitrator will give priority to clauses/reservation stations closer to the bottom of the pipeline. It will not execute an alu clause until the fetch fetches initiated by the previous fetch clause have completed. There are two separate sets of reservation stations, one for pixel vectors and one for vertices vectors. This way a pixel can pass a vertex and a vertex can pass a pixel.

To support the shader pipe the sequencer also contains the shader instruction cache, constant store, control flow constants and texture state. The four shader pipes also execute the same instruction thus there is only one sequencer for the whole chip.

The sequencer first arbitrates between vectors of 64 vertices that arrive directly from primitive assembly and vectors of 16 quads (64 pixels) that are generated in the scan converter.

The vertex or pixel program specifies how many GPRs it needs to execute. The sequencer will not start the next vector until the needed space is available in the GPRs.

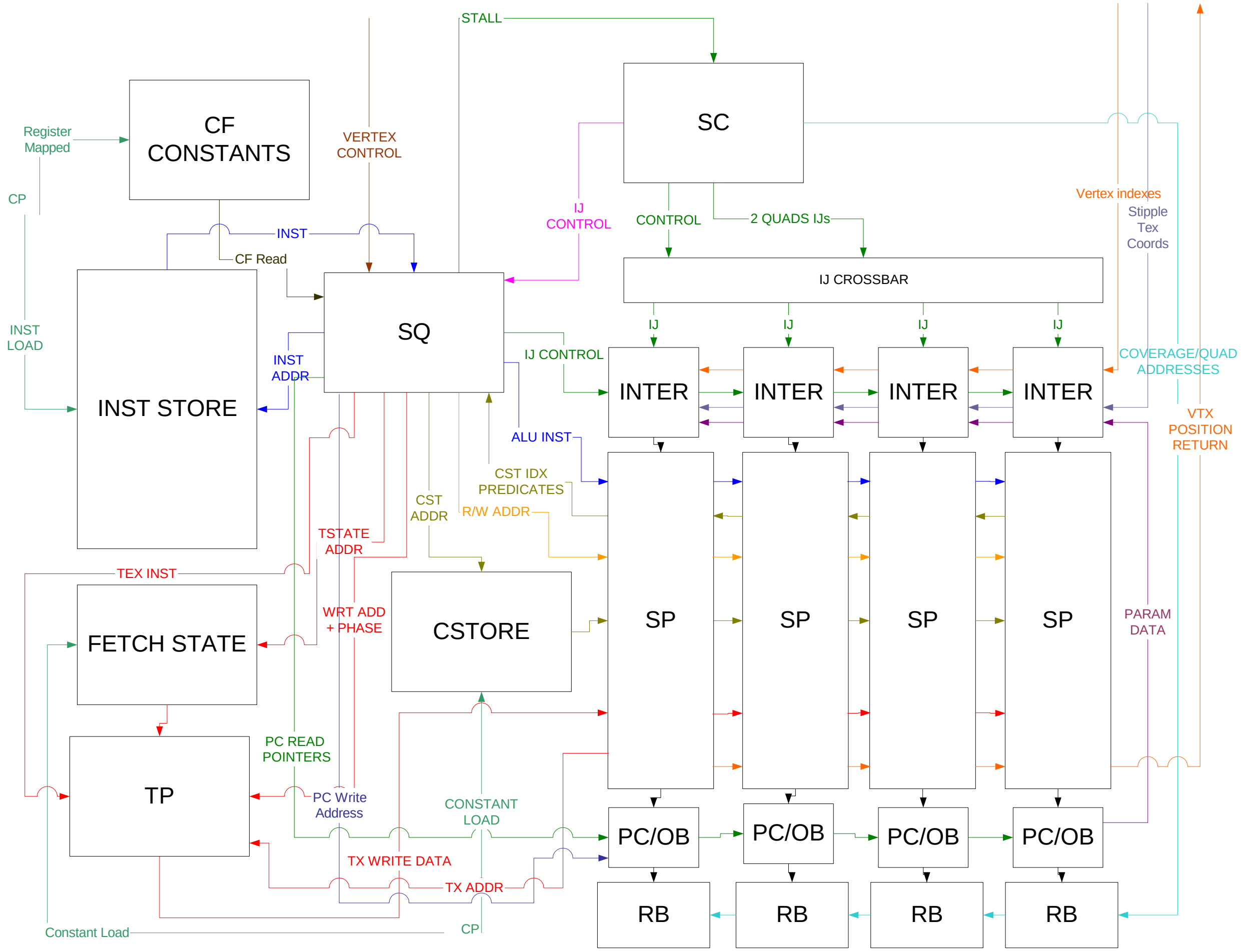


Figure 1: General Sequencer overview

1.1 Top Level Block Diagram

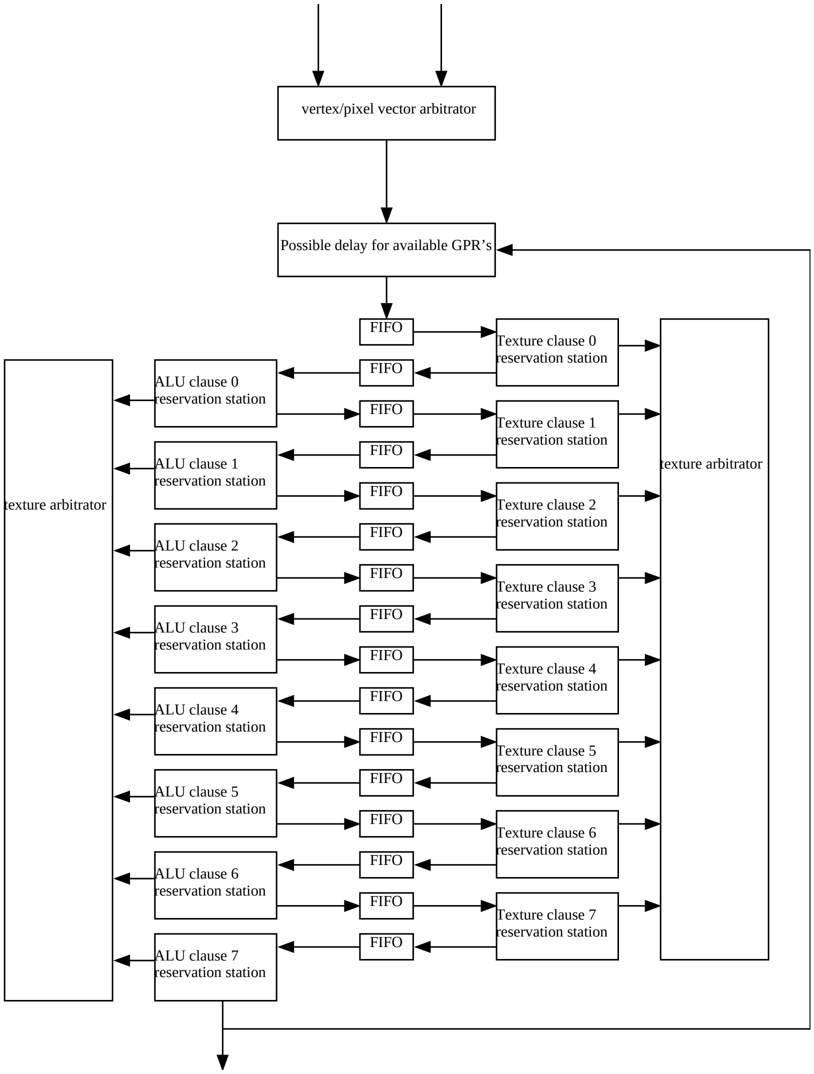

vertex/pixel vector arbitrator
Possible delay for available GPR's
FIFO
Texture clause 0 reservation station
Texture clause 1 reservation station
Texture clause 2 reservation station
Texture clause 3 reservation station
Texture clause 4 reservation station
Texture clause 5 reservation station
Texture clause 6 reservation station
Texture clause 7 reservation station
ALU clause 0 reservation station
ALU clause 1 reservation station
ALU clause 2 reservation station
ALU clause 3 reservation station
ALU clause 4 reservation station
ALU clause 5 reservation station
ALU clause 6 reservation station
ALU clause 7 reservation station
texture arbitrator
texture arbitrator

Figure 2: Reservation stations and arbiters

There are two sets of the above figure, one for vertices and one for pixels.

Depending on the arbitration state, the sequencer will either choose a vertex or a pixel packet. The control packet consists of 3 bits of state, 7 bits for the base address of the Shader program and some information on the coverage to determine fetch LOD plus other various small state bits.

On receipt of a packet, the input state machine (not pictured but just before the first FIFO) allocated enough space in the GPRs to store the interpolated values and temporaries. Following this, the barycentric coordinates (and XY screen position if needed) are sent to the interpolator, which will use them to interpolate the parameters and place the results into the GPRs. Then, the input state machine stacks the packet in the first FIFO.

On receipt of a command, the level 0 fetch machine issues a fetch request to the TP and corresponding GPR address for the fetch address (ta). A small command (tcmd) is passed to the fetch system identifying the current level number (0) as well as the GPR write address for the fetch return data. One fetch request is sent every 4 clocks causing the texturing of sixteen 2x2s worth of data (or 64 vertices). Once all the requests are sent the packet is put in FIFO 1.

Upon receipt of the return data, the fetch unit writes the data to the register file using the write address that was provided by the level 0 fetch machine and sends the clause number (0) to the level 0 fetch state machine to signify that the write is done and thus the data is ready. Then, the level 0 fetch machine increments the counter of FIFO 1 to signify to the ALU 0 that the data is ready to be processed.

On receipt of a command, the level 0 ALU machine first decrements the input FIFO 1 counter and then issues a complete set of level 0 shader instructions. For each instruction, the ALU state machine generates 3 source addresses, one destination address and an instruction. Once the last instruction has been issued, the packet is put into FIFO 2.

There will always be two active ALU clauses at any given time (and two arbiters). One arbiter will arbitrate over the odd instructions (4 clocks cycles) and the other one will arbitrate over the even instructions (4 clocks cycles). The only constraints between the two arbiters is that they are not allowed to pick the same clause number as the other one is currently working on if the packet is not of the same type (render state).

If the packet is a vertex packet, upon reaching ALU clause 3, it can export the position if the position is ready. So the arbiter must prevent ALU clause 3 to be selected if the positional buffer is full (or can't be accessed). Along with the positional data, if needed the sprite size and/or edge flags can also be sent.

A special case is for multipass vertex shaders, which can export 12 parameters per last 6 clauses to the output buffer. If the output buffer is full or doesn't have enough space the sequencer will prevent such a vertex group to enter an exporting clause.

Multipass pixel shaders can export 12 parameters to memory from the last clause only (7).

All other clauses process in the same way until the packet finally reaches the last ALU machine (7).

Only one pair of interleaved ALU state machines may have access to the register file address bus or the instruction decode bus at one time. Similarly, only one fetch state machine may have access to the register file address bus at one time. Arbitration is performed by three arbiter blocks (two for the ALU state machines and one for the fetch state machines). The arbiters always favor the higher number state machines, preventing a bunch of half finished jobs from clogging up the register files.

1.2 Data Flow graph (SP)

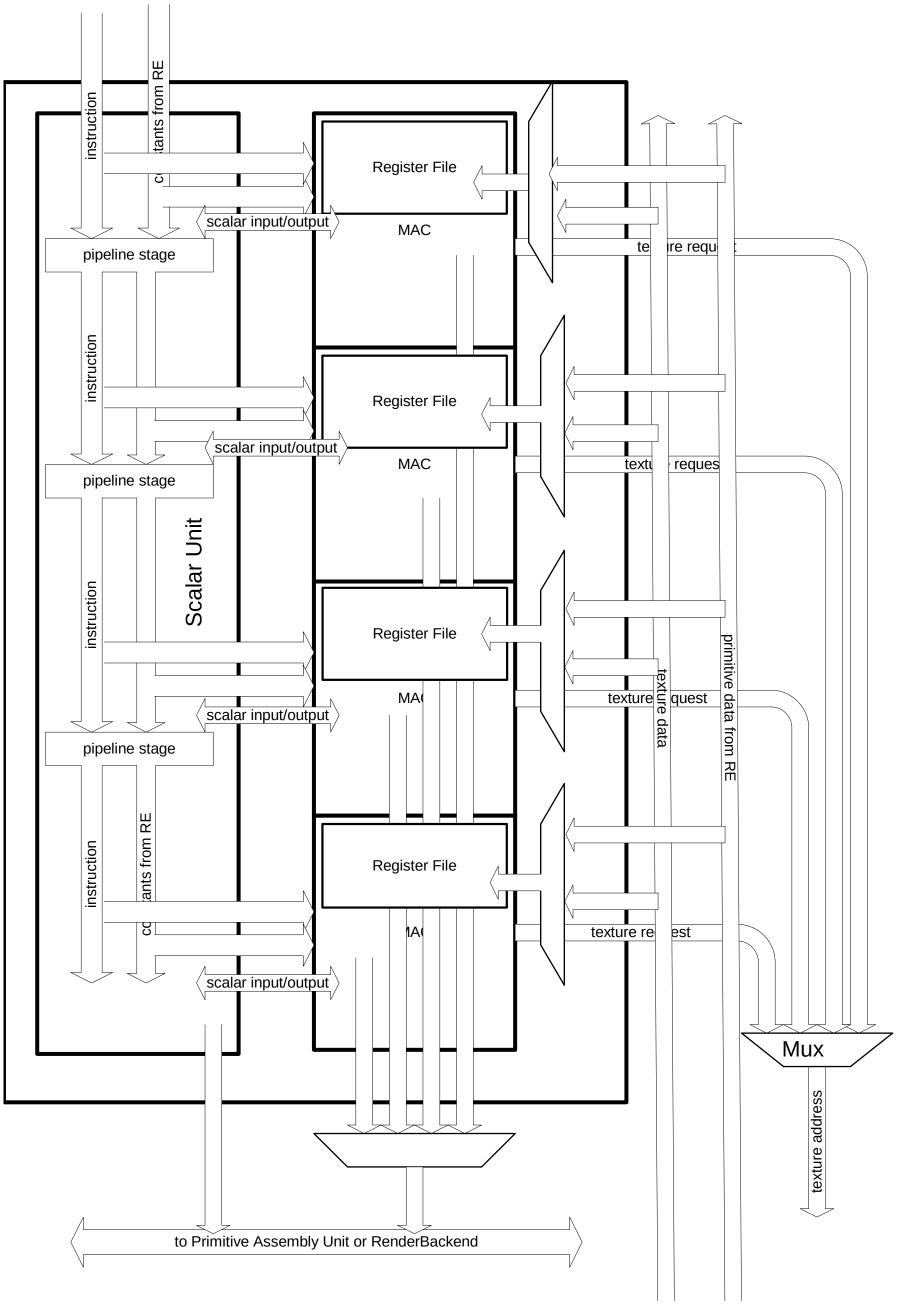


Figure 3: The shader Pipe

The gray area represents blocks that are replicated 4 times per shader pipe (16 times on the overall chip).

1.3 Control Graph

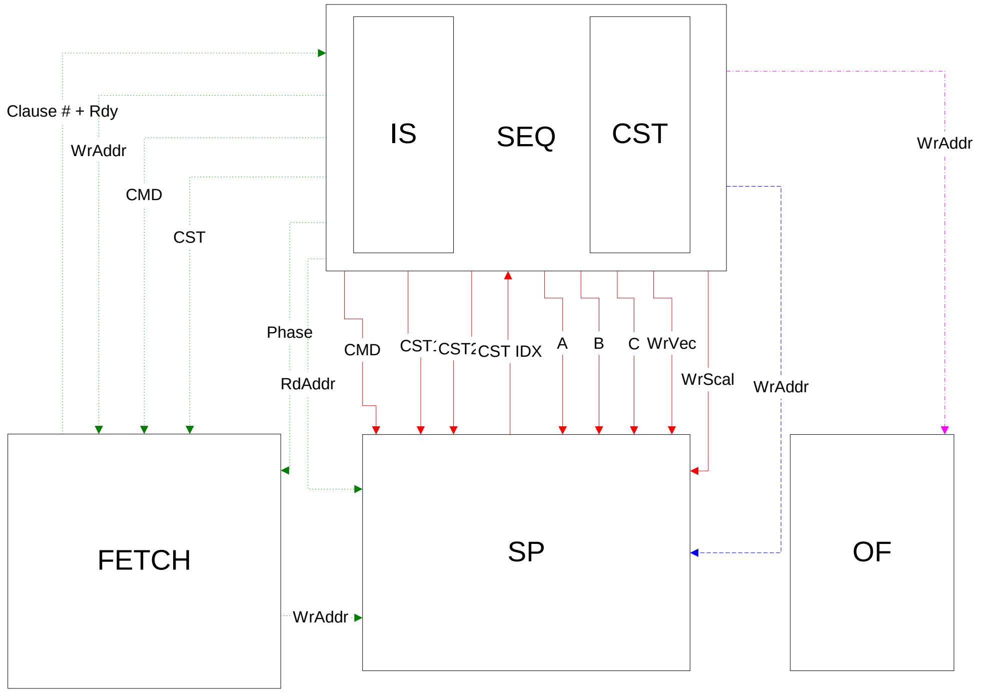


Figure 4: Sequencer Control interfaces

In green is represented the Fetch control interface, in red the ALU control interface, in blue the Interpolated/Vector control interface and in purple is the output file control interface.

2. Interpolated data bus

The interpolators contain an IJ buffer to pack the information as much as possible before writing it to the register file.

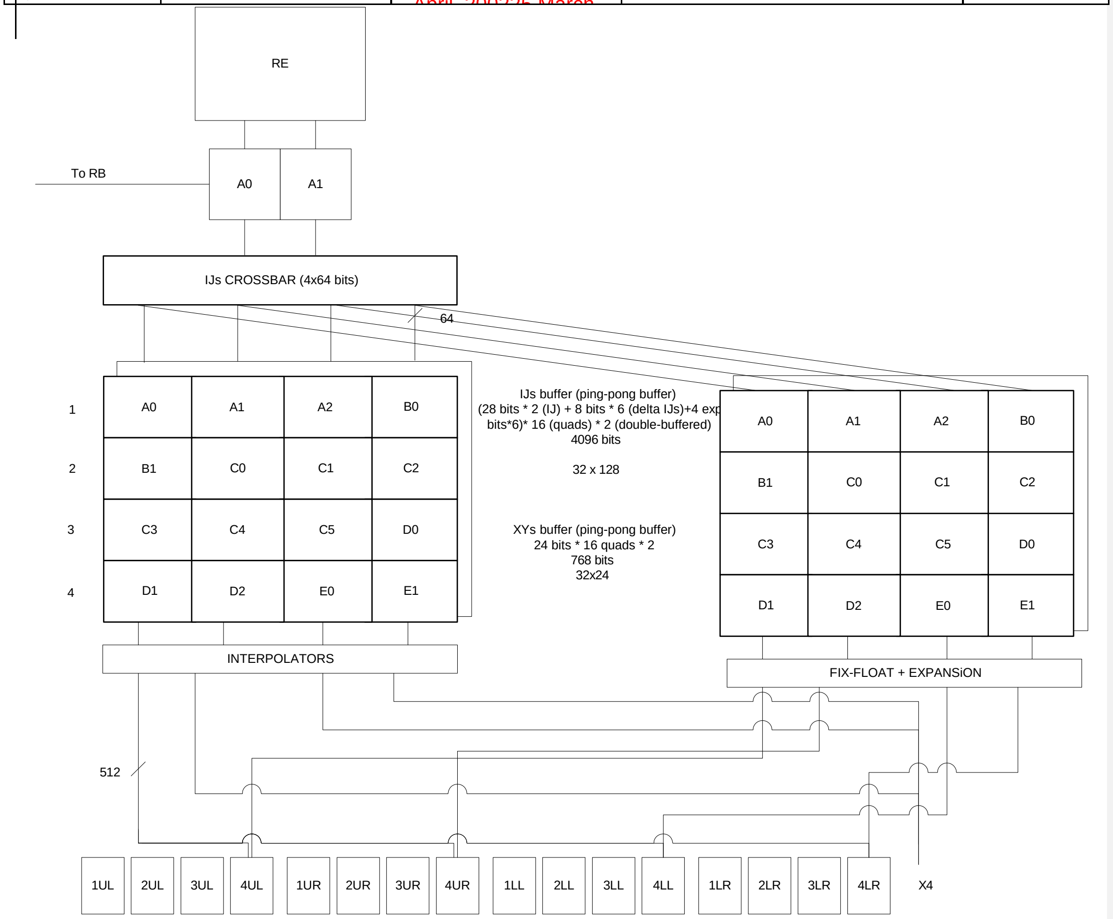


Figure 5: Interpolation buffers

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21	T22	T23
											WRITES													
SP 0	A0	A0	XY A0	B1	B1	XY B1	C3	C3	XY C3				D1	D1	XY D1									
SP 1	A1	A1	XY A1				C0	C0	XY C0	C4	C4	XY C4	D2	D2	XY D2									
SP 2	A2	A2	XY A2				C1	C1	XY C1	C5	C5	XY C5				E0	E0	XY E0						
SP 3				B0	B0	XY B0	C2	C2	XY C2				D0	D0	XY D0	E1	E1	XY E1						
											READS													
SP 0	XY 0-3	XY 16-19	XY 32-35	XY 48-51	A0	B1	C3	D1					A0	B1	C3	D1					V 0-3	V 16-19	V 32-35	V 48-51
SP 1	XY 4-7	XY 20-23	XY 36-39	XY 52-55	A1		C4	D2		C0			A1		C4	D2		C0			V 4-7	V 20-23	V 36-39	V 52-55
SP 2	XY 8-11	XY 24-27	XY 40-43	XY 56-59	A2		C5			C1		E0	A2		C5			C1		E0	V 8-11	V 24-27	V 40-43	V 56-59
SP 3	XY 12-15	XY 28-31	XY 44-47	XY 60-63					B0	C2	D0	E1					B0	C2	D0	E1	V 12-15	V 28-31	V 44-47	V 60-63
	XY				P1								P2								VTX			

Figure 6: Interpolation timing diagram

Above is an example of a tile the sequencer might receive from the SC. The write side is how the data get stacked into the XY and IJ buffers, the read side is how the data is passed to the GPRs. The IJ information is packed in the IJ buffer 4 quads at a time or two clocks. The sequencer allows at any given time as many as four quads to interpolate a parameter. They all have to come from the same primitive. Then the sequencer controls the write mask to the GPRs to write the valid data in.

3. Instruction Store

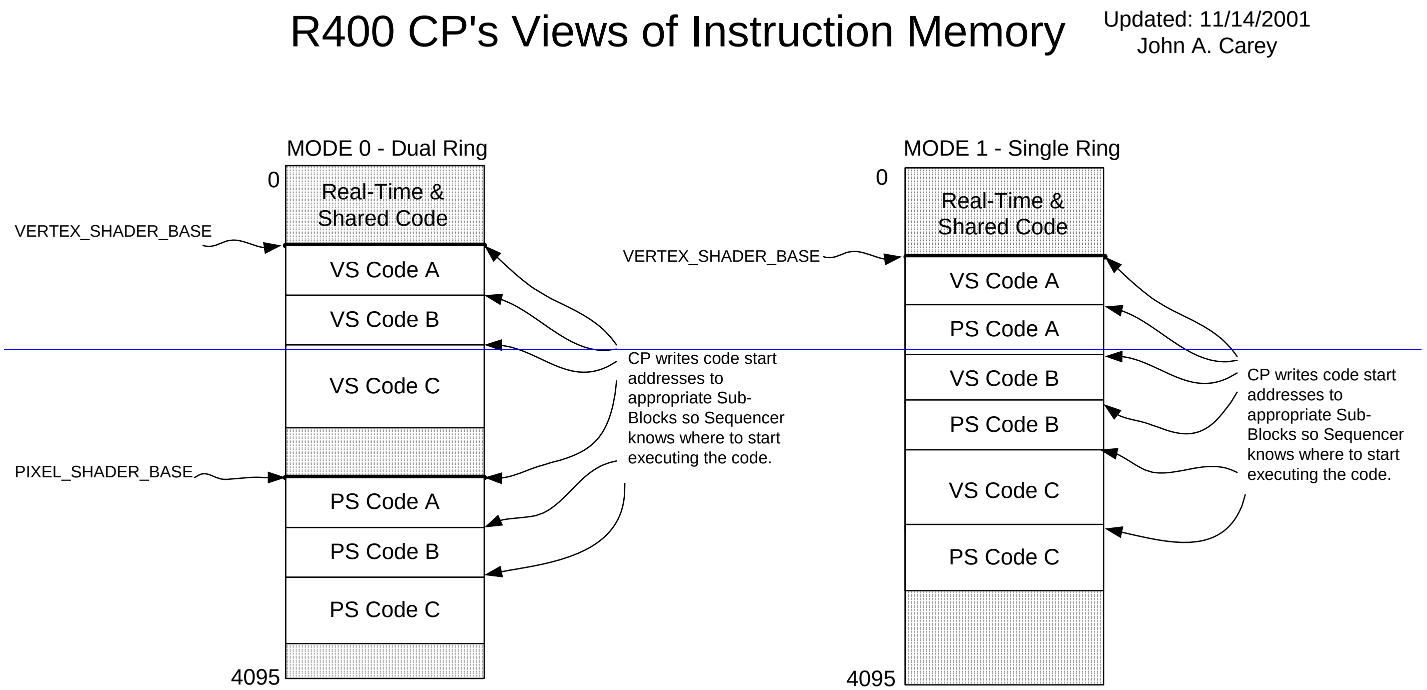
There is going to be only one instruction store for the whole chip. It will contain 4096 instructions of 96 bits each.

It is likely to be a 1 port memory; we use 1 clock to load the ALU instruction, 1 clocks to load the Fetch instruction, 1 clock to load 2 control flow instructions and 1 clock to write instructions.

The instruction store is loaded by the CP thru the register mapped registers.

~~The next picture shows the various modes the CP can load the memory. The Sequencer has to keep track of the loading modes in order to wrap around the correct boundaries. The wrap-around points are arbitrary and they are specified in the VS_BASE and PIX_BASE control registers.~~ The VS_BASE and PS_BASE context registers are used to specify for each context where its shader is in the instruction memory.

For the Real time commands the story is quite the same but for some small differences. There are no wrap-around points for real time so the driver must be careful not to overwrite regular shader data. The shared code (shared subroutines) uses the same path as real time.



4. Figure 7: The CP's view of the instruction memory

Formatted: Bullets and Numbering

~~5.~~4. Sequencer Instructions

All control flow instructions and move instructions are handled by the sequencer only. The ALUs will perform NOPs during this time (MOV PV,PV, PS,PS) if they have nothing else to do.

~~6.~~5. Constant Stores

~~6.1~~5.1 Memory organizations

A likely size for the ALU constant store is 1024x128 bits. The read BW from the ALU constant store is 128 bits/clock and the write bandwidth is 32 bits/clock (directed by the CP bus size not by memory ports).

The maximum logical size of the constant store for a given shader is 256 constants. Or 512 for the pixel/vertex shader pair. The size of the re-mapping table is 128 lines (each line addresses 4 constants). The write granularity is 4 constants or 512 bits. It takes 16 clocks to write the four constants. Real time requires 256 lines in the physical memory (this is physically register mapped).

The texture state is also kept in a similar memory. The size of this memory is 320x96 bits (128 texture states for regular mode, 32 states for RT). The memory thus holds 128 texture states (192 bits per state). The logical size exposes 32 different states total, which are going to be shared between the pixel and the vertex shader. The size of the re-mapping table to for the texture state memory is 32 lines (each line addresses 1 texture state lines in the real memory). The CP write granularity is 1 texture state lines (or 192 bits). The driver sends 512 bits but the CP ignores the top 320 bits. It thus takes 6 clocks to write the texture state. Real time requires 32 lines in the physical memory (this is physically register mapped).

The control flow constant memory doesn't sit behind a renaming table. It is register mapped and thus the driver must reload its content each time there is a change in the control flow constants. Its size is 320*32 because it must hold 8 copies of the 32 dwords of control flow constants and the loop construct constants must be aligned.

The constant re-mapping tables for texture state and ALU constants are logically register mapped for regular mode and physically register mapped for RT operation.

~~6.2~~5.2 Management of the Control Flow Constants

The control flow constants are register mapped, thus the CP writes to the according register to set the constant, the SQ decodes the address and writes to the block pointed by its current base pointer (CF_WR_BASE). On the read side, one level of indirection is used. A register (SQ_CONTEXT_MISC.CF_RD_BASE) keeps the current base pointer to the control flow block. This register is copied whenever there is a state change. Should the CP write to CF after the state change, the base register is updated with the (current pointer number +1)% number of states. This way, if the CP doesn't write to CF the state is going to use the previous CF constants.

~~6.3~~5.3 Management of the re-mapping tables

~~6.3.1~~5.3.1 *R400 Constant management*

The sequencer is responsible to manage two re-mapping tables (one for the constant store and one for the texture state). On a state change (by the driver), the sequencer will broadside copy the contents of its re-mapping tables to a new one. We have 8 different re-mapping tables we can use concurrently.

The constant memory update will be incremental, the driver only need to update the constants that actually changed between the two state changes.

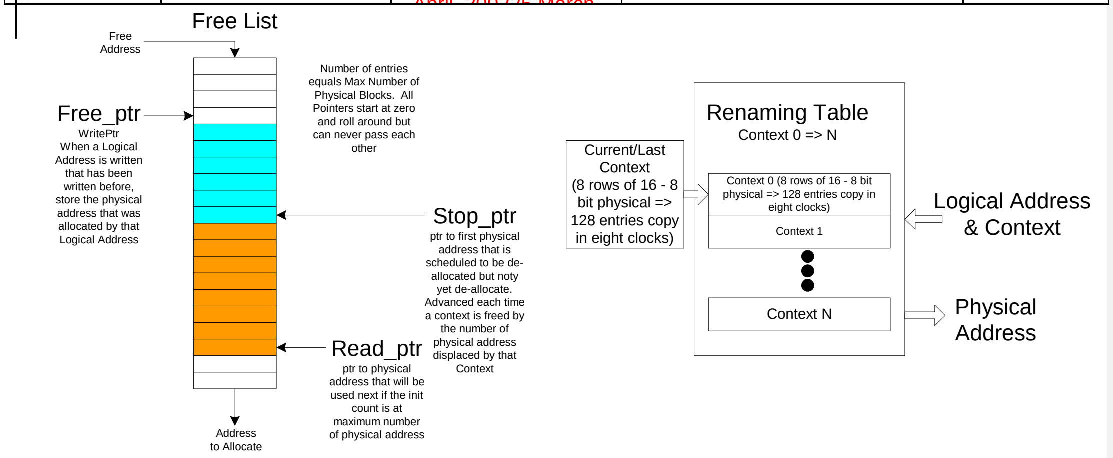
For this model to work in its simplest form, the requirement is that the physical memory MUST be at least twice as large as the logical address space + the space allocated for Real Time. In our case, since the logical address space

is 512 and the reserved RT space can be up to 256 entries, the memory must be of sizes 1280 and above. Similarly the size of the texture store must be of 32*2+32 = 96 entries and above.

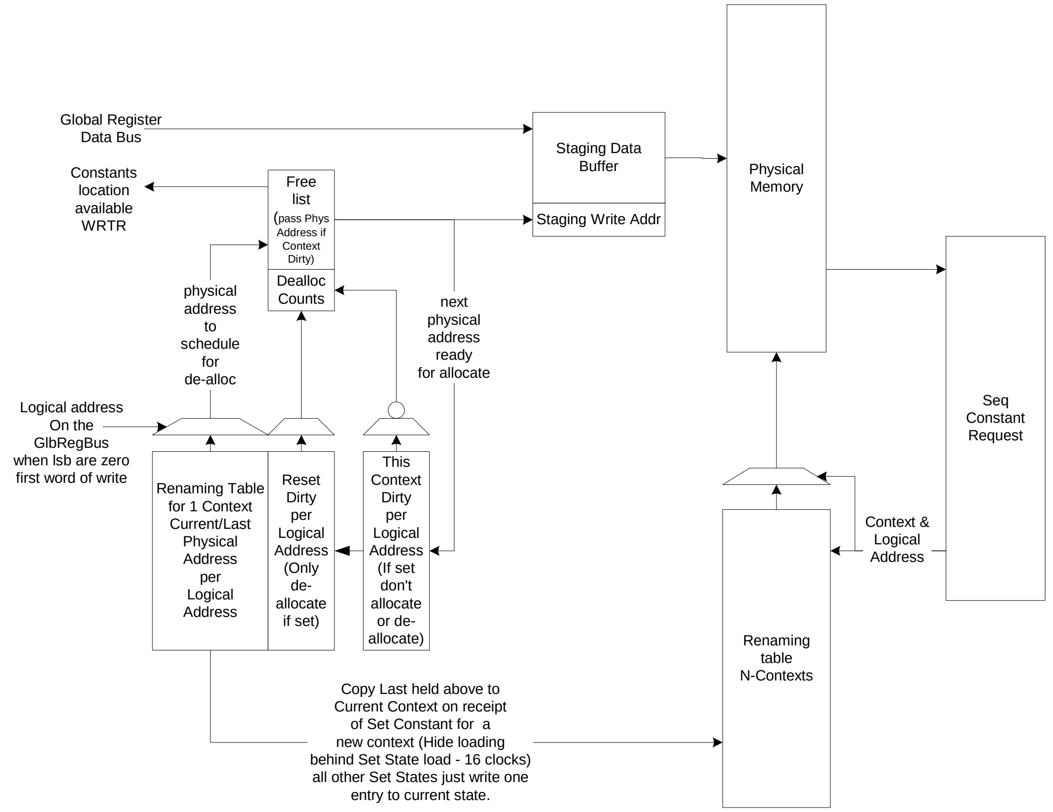
~~6.3.2~~5.3.2 Proposal for R400LE constant management

To make this scheme work with only 512+256 = 768 entries, upon reception of a CONTROL packet of state + 1, the sequencer would check for SQ_IDLE and PA_IDLE and if both are idle will erase the content of state to replace it with the new state (this is depicted in Figure 8: De-allocation mechanism~~Figure 9: De-allocation mechanism~~~~Figure 9: De-allocation mechanism~~). Note that in the case a state is cleared a value of 0 is written to the corresponding de-allocation counter location so that when the SQ is going to report a state change, nothing will be de-allocated upon the first report.

The second path sets all context dirty bits that were used in the current state to 1 (thus allowing the new state to reuse these physical addresses if needed).



Free List
Free Address
Free_ptr
WritePtr
When a Logical Address is written that has been written before, store the physical address that was allocated by that Logical Address
Number of entries equals Max Number of Physical Blocks. All Pointers start at zero and roll around but can never pass each other
Stop_ptr
ptr to first physical address that is scheduled to be de-allocated but noty yet de-allocate. Advanced each time a context is freed by the number of physical address displaced by that Context
Read_ptr
ptr to physical address that will be used next if the init count is at maximum number of physical address
Address to Allocate
Current/Last Context (8 rows of 16 - 8 bit physical => 128 entries copy in eight clocks)
Renaming Table
Context 0 => N
Context 0 (8 rows of 16 - 8 bit physical => 128 entries copy in eight clocks)
Context 1
Context N
Logical Address & Context
Physical Address



Global Register Data Bus
Staging Data Buffer
Staging Write Addr
Physical Memory
Constants location available WRTR
Free list (pass Phys Address if Context Dirty)
Dealloc Counts
physical address to schedule for de-alloc
next physical address ready for allocate
Seq Constant Request
Logical address On the GlbRegBus when lsb are zero first word of write
Renaming Table for 1 Context Current/Last Physical Address per Logical Address
Reset Dirty per Logical Address (Only de-allocate if set)
This Context Dirty per Logical Address (If set don't allocate or de-allocate)
Context & Logical Address
Renaming table N-Contexts
Copy Last held above to Current Context on receipt of Set Constant for a new context (Hide loading behind Set State load - 16 clocks) all other Set States just write one entry to current state.

Figure 7~~8~~: Constant management

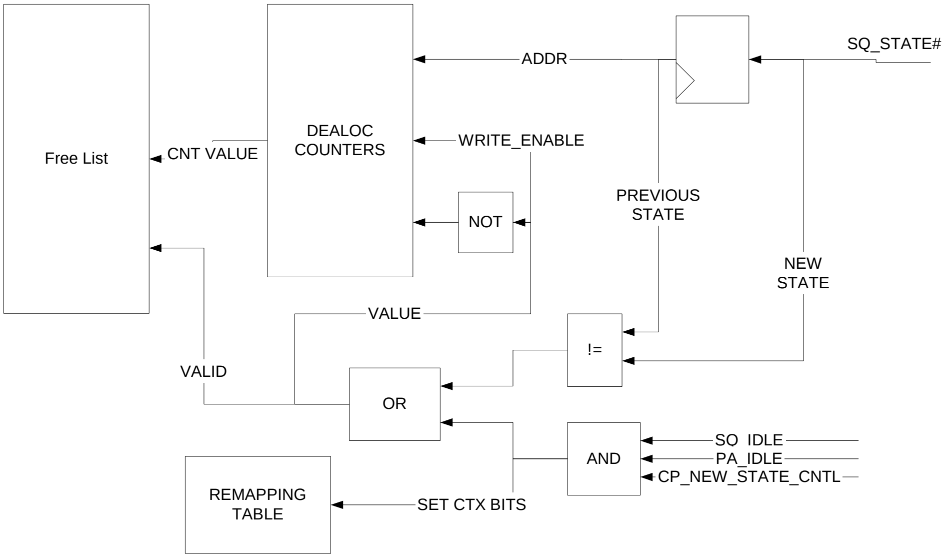


Figure ~~9~~8: De-allocation mechanism for R400LE

~~6.3.3~~5.3.3 *Dirty bits*

Two sets of dirty bits will be maintained per logical address. The first one will be set to zero on reset and set when the logical address is addressed. The second one will be set to zero whenever a new context is written and set for each address written while in this context. The reset dirty is not set, then writing to that logical address will not require de-allocation of whatever address stored in the renaming table. If it is set and the context dirty is not set, then the physical address store needs to be de-allocated and a new physical address is necessary to store the incoming data. If they are both set, then the data will be written into the physical address held in the renaming for the current logical address. No de-allocation or allocation takes place. This will happen when the driver does a set constant twice to the same logical address between context changes. NOTE: It is important to detect and prevent this, failure to do it will allow multiple writes to allocate all physical memory and thus hang because a context will not fit for rendering to start and thus free up space.

~~6.3.4~~5.3.4 *Free List Block*

A free list block that would consist of a counter (called the IFC or Initial Free Counter) that would reset to zero and incremented every time a chunk of physical memory is used until they have all been used once. This counter would be checked each time a physical block is needed, and if the original ones have not been used up, us a new one, else check the free list for an available physical block address. The count is the physical address for when getting a chunk from the counter.

Storage of a free list big enough to store all physical block addresses.

Maintain three pointers for the free list that are reset to zero. The first one we will call write_ptr. This pointer will identify the next location to write the physical address of a block to be de-allocated. Note: we can never free more physical memory locations than we have. Once recording address the pointer will be incremented to walk the free list like a ring.

The second pointer will be called stop_ptr. The stop_ptr pointer will be advanced by the number of address chunks de-allocates when a context finishes. The address between the stop_ptr and write_ptr cannot be reused because they are still in use. But as soon as the context using then is dismissed the stop_ptr will be advanced.

The third pointer will be called read_ptr. This pointer will point will point to the next address that can be used for allocation as long as the read_ptr does not equal the stop_ptr and the IFC is at its maximum count.

~~6.3.5~~5.3.5 *De-allocate Block*

This block will maintain a free physical address block count for each context. While in current context, a count shall be maintained specifying how many blocks were written into the free list at the write_ptr pointer. This count will be reset upon reset or when this context is active on the back and different than the previous context. It is actually a count of blocks in the previous context that will no longer be used. This count will be used to advance the write_ptr pointer to make available the set of physical blocks freed when the previous context was done. This allows the discard or de-allocation of any number of blocks in one clock.

~~6.3.6~~5.3.6 *Operation of Incremental model*

The basic operation of the model would start with the write_ptr, stop_ptr, read_ptr pointers in the free list set to zero and the free list counter is set to zero. Also all the dirty bits and the previous context will be initialized to zero. When the first set constants happen, the reset dirty bit will not be set, so we will allocate a physical location from the free list counter because its not at the max value. The data will be written into physical address zero. Both the additional copy of the renaming table and the context zeros of the big renaming table will be updated for the logical address that was written by set start with physical address of 0. This process will be repeated for any logical address that are not dirty until the context changes. If a logical address is hit that has its dirty bits set while in the same context, both dirty bits would be set, so the new data will be over-written to the last physical address assigned for this logical address. When the first draw command of the context is detected, the previous context stored in the additional renaming table will be copied to the larger renaming table in the current (new) context location. Then the set constant logical address with be loaded with a new physical address during the copy and if the reset dirty was set, the physical address it replaced in the renaming table would be entered at the write_ptr pointer location on the free list and the write_ptr will be incremented. The de-allocation counter for the previous context (eight) will be incremented. This as set states come in for this context one of the following will happen:

1.) No dirty bits are set for the logical address being updated. A line will be allocated of the free-list counter or the free list at read_ptr pointer if read_ptr != to stop_ptr .
2.) Reset dirty set and Context dirty not set. A new physical address is allocated, the physical address in the renaming table is put on the free list at write_ptr and it is incremented along with the de-allocate counter for the last context.
3.) Context dirty is set then the data will be written into the physical address specified by the logical address.

This process will continue as long as set states arrive. This block will provide backpressure to the CP whenever he has not free list entries available (counter at max and stop_ptr == read_ptr). The command stream will keep a count of contexts of constants in use and prevent more than max constants contexts from being sent.

Whenever a draw packet arrives, the content of the re-mapping table is written to the correct re-mapping table for the context number. Also if the next context uses less constants than the current one all exceeding lines are moved to the free list to be de-allocated later. This happens in parallel with the writing of the re-mapping table to the correct memory.

Now preferable when the constant context leaves the last ALU clause it will be sent to this block and compared with the previous context that left. (Init to zero) If they differ than the older context will no longer be referenced and thus can be de-allocated in the physical memory. This is accomplished by adding the number of blocks freed this context to the stop_ptr pointer. This will make all the physical addresses used by this context available to the read_ptr allocate pointer for future allocation.

This device allows representation of multiple contexts of constants data with N copies of the logical address space. It also allows the second context to be represented as the first set plus some new additional data by just storing the delta's. It allows memory to be efficiently used and when the constants updates are small it can store multiple context. However, if the updates are large, less contexts will be stored and potentially performance will be degraded. Although it will still perform as well as a ring could in this case.

~~6.4~~5.4 Constant Store Indexing

In order to do constant store indexing, the sequencer must be loaded first with the indexes (that come from the GPRs). There are 144 wires from the exit of the SP to the sequencer (9 bits pointers x 16 vertexes/clock). Since the data must pass thru the Shader pipe for the float to fixed conversion, there is a latency of 4 clocks (1 instruction)

between the time the sequencer is loaded and the time one can index into the constant store. The assembly will look like this

```
MOVA  R1.X,R2.X        // Loads the sequencer with the content of R2.X, also copies the content of R2.X into R1.X
NOP                    // latency of the float to fixed conversion
ADD   R3,R4,C0[R2.X]// Uses the state from the sequencer to add R4 to C0[R2.X] into R3
```

Note that we don't really care about what is in the brackets because we use the state from the MOVA instruction. R2.X is just written again for the sake of simplicity and coherency.

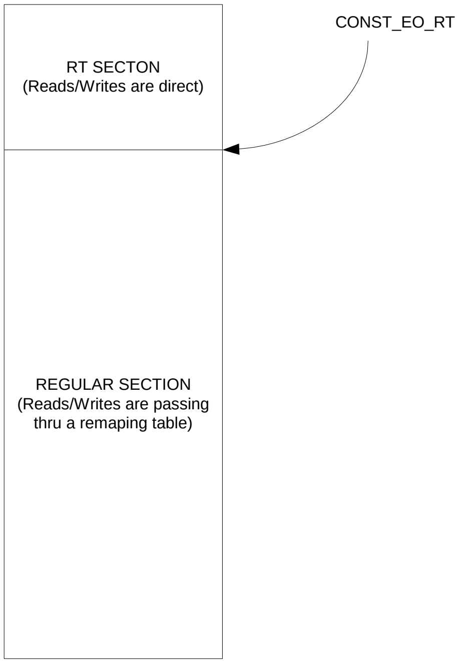
The storage needed in the sequencer in order to support this feature is 2*64*9 bits = 1152 bits.

~~6.5~~5.5 Real Time Commands

The real time commands constants are written by the CP using the register mapped registers allocated for RT. It works is the same way than when dealing with regular constant loads BUT in this case the CP is not sending a logical address but rather a physical address and the reads are not passing thru the re-mapping table but are directly read from the memory. The boundary between the two zones is defined by the CONST_EO_RT control register. Similarly, for the fetch state, the boundary between the two zones is defined by the TSTATE_EO_RT control register.

~~6.6~~5.6 Constant Waterfalling

In order to have a reasonable performance in the case of constant store indexing using the address register, we are going to have the possibility of using the physical memory port for read only. This way we can read 1 constant per clock and thus have a worst-case waterfall mode of 1 vertex per clock. There is a small synchronization issue related with this as we need for the SQ to make sure that the constants where actually written to memory (not only sent to the sequencer) before it can allow the first vector of pixels or vertices of the state to go thru the ALUs. To do so, the sequencer keeps 8 bits (one per render state) and sets the bits whenever the last render state is written to memory and clears the bit whenever a state is freed.



CONST_EO_RT
RT SECTON
(Reads/Writes are direct)
REGULAR SECTION
(Reads/Writes are passing thru a remaping table)

Figure 9~~10~~: The instruction store

~~7.~~6. Looping and Branches

Loops and branches are planned to be supported and will have to be dealt with at the sequencer level. We plan on supporting constant loops and branches using a control program.

~~7.1~~6.1 The controlling state.

The R400 controling state consists of:

Boolean[256:0]
Loop_count[7:0][31:0]
Loop_Start[7:0][31:0]
Loop_Step[7:0][31:0]

That is 256 Booleans and 32 loops.

We have a stack of 4 elements for nested calls of subroutines and 4 loop counters to allow for nested loops.

This state is available on a per shader program basis.

~~7.2~~6.2 The Control Flow Program

Examples of control flow programs are located in the R400 programming guide document.

The basic model is as follows:

The render state defined the clause boundaries:
Vertex_shader_fetch[7:0][7:0]   // eight 8 bit pointers to the location where each clauses control program is located
Vertex_shader_alu[7:0][7:0]   // eight 8 bit pointers to the location where each clauses control program is located
Pixel_shader_fetch[7:0][7:0]   // eight 8 bit pointers to the location where each clauses control program is located
Pixel_shader_alu[7:0][7:0]   // eight 8 bit pointers to the location where each clauses control program is located

A pointer value of FF means that the clause doesn't contain any instructions.

The control program for a given clause is executed to completion before moving to another clause, (with the exception of the pick two nature of the alu execution). The control program is the only program aware of the clause boundaries.

The control program has nine basic instructions:

Execute
Conditional_execute
Conditional_Execute_Predicates
Conditional_jump
Conditionnal_Call
Return
Loop_start
Loop_end
NOP

Execute, causes the specified number of instructions in instruction store to be executed.
Conditional_execute checks a condition first, and if true, causes the specified number of instructions in instruction store to be executed.
Loop_start resets the corresponding loop counter to the start value on the first pass after it checks for the end condition and if met jumps over to a specified address.
Loop_end increments (decrements?) the loop counter and jumps back the specified number of instructions.
Conditionnal_Call jumps to an address and pushes the IP counter on the stack if the condition is met. On the return instruction, the IP is popped from the stack.

Conditional_execute_Predicates executes a block of instructions if all bits in the predicate vectors meet the condition.
Conditional_jumps jumps to an address if the condition is met.
NOP is a regular NOP

NOTE THAT ALL JUMPS MUST JUMP TO EVEN CFP ADDRESSES since there are two control flow instructions per memory line. Thus the compiler must insert NOPs where needed to align the jumps on even CFP addresses.

Also if the jump is logically bigger than pshader_cntl_size (or vshader_cntl_size) we break the program (clause) and set the debug registers. If an execute or conditional_execute is lower than cntl_size or bigger than size we also break the program (clause) and set the debug registers.

We have to fit instructions into 48 bits in order to be able to put two control flow instruction per line in the instruction store.

A value of 1 in the Addressing means that the address specified in the Exec Address field (or in the jump address field) is an ABSOLUTE address. If the addressing field is cleared (should be the default) then the address is relative to the base of the current shader program.

Note that whenever a field is marked as RESERVED, it is assumed that all the bits of the field are cleared (0).

Execute					
47	46… 42	41	40 … 24	23 … 12	11 … 0
Addressing	00001	Last	RESERVED	Instruction count	Exec Address

Execute up to 4k instructions at the specified address in the instruction memory. If Last is set, this is the last group of instructions of the clause.

NOP			
47	46 … 42	41	40 … 0
Addressing	00010	Last	RESERVED

This is a regular NOP. If Last is set, this is the last instruction of the clause.

Conditional_Execute								
47	46 … 42	41	40	39 … 32	31	30 … 24	23 … 12	11 … 0
Addressing	00011	Last	RESERVED	Boolean address	Condition	RESERVED	Instruction count	Exec Address

If the specified Boolean (8 bits can address 256 Booleans) meets the specified condition then execute the specified instructions (up to 4k instructions). If Last is set, then if the condition is met, this is the last group of instructions to be executed in the clause. If the condition is not met, we go on to the next control flow instruction.

Conditional_Execute_Predicates								
47	46 … 42	41	40 … 34	33 … 32	31	30 … 24	23 … 12	11 … 0
Addressing	00100	Last	RESERVED	Predicate vector	Condition	RESERVED	Instruction count	Exec Address

Check the AND/OR of all current predicate bits. If AND/OR matches the condition execute the specified number of instructions. We need to AND/OR this with the kill mask in order not to consider the pixels that aren't valid. If Last is set, then if the condition is met, this is the last group of instructions to be executed in the clause. If the condition is not met, we go on to the next control flow instruction.

Loop_Start				
47	46 … 42	41 … 17	16 … 12	11 … 0
Addressing	00101	RESERVED	loop ID	Jump address

Loop Start. Compares the loop iterator with the end value. If loop condition not met jump to the address. Forward jump only. Also computes the index value. The loop id must match between the start to end, and also indicates which control flow constants should be used with the loop.

Loop_End				
47	46 … 42	41 … 17	16 … 12	11 … 0
Addressing	00110	RESERVED	loop ID	start address

Loop end. Increments the counter by one, compares the loop count with the end value. If loop condition met, continue, else, jump BACK to the start of the loop.

The way this is described does not prevent nested loops, and the inclusion of the loop id make this easy to do.

Conditionnal_Call						
47	46 … 42	41 … 34	33 … 32	31	30 … 12	11 … 0
Addressing	00111	RESERVED	Predicate vector	Condition	RESERVED	Jump address

If the condition is met, jumps to the specified address and pushes the control flow program counter on the stack.

Return		
47	46 … 42	41 … 0
Addressing	01000	RESERVED

Pops the topmost address from the stack and jumps to that address. If nothing is on the stack, the program will just continue to the next instruction.

Conditionnal_Jump							
47	46 … 42	41 … 40	39 … 32	31	30	29 … 12	11 … 0
Addressing	01001	RESERVED	Boolean address	Condition	FW only	RESERVED	Jump address

If condition met, jumps to the address. FORWARD jump only allowed if bit 31 set. Bit 31 is only an optimization for the compiler and should NOT be exposed to the API.

To prevent infinite loops, we will keep 9 bits loop iterators instead of 8 (we are only able to loop 256 times). If the counter goes higher than 255 then the loop_end or the loop_start instruction is going to break the loop and set the debug GPRs.

~~7.3~~6.3 Data dependant predicate instructions

Data dependant conditionals will be supported in the R400. The only way we plan to support those is by supporting three vector/scalar predicate operations of the form:

PRED_SETE_# - similar to SETE except that the result is 'exported' to the sequencer.
PRED_SETNE_# - similar to SETNE except that the result is 'exported' to the sequencer.
PRED_SETGT_# - similar to SETGT except that the result is 'exported' to the sequencer
PRED_SETGTE_# - similar to SETGTE except that the result is 'exported' to the sequencer

For the scalar operations only we will also support the two following instructions:
PRED_SETE0_# – SETE0
PRED_SETE1_# – SETE1

The export is a single bit - 1 or 0 that is sent using the same data path as the MOVA instruction. The sequencer will maintain 4 sets of 64 bit predicate vectors (in fact 8 sets because we interleave two programs but only 4 will be exposed) and use it to control the write masking. This predicate is not maintained across clause boundaries. The # sign is used to specify which predicate set you want to use 0 thru 3.

Then we have two conditional execute bits. The first bit is a conditional execute "on" bit and the second bit tells us if we execute on 1 or 0. For example, the instruction:

P0_ADD_# R0,R1,R2

Is only going to write the result of the ADD into those GPRs whose predicate bit is 0. Alternatively, P1_ADD_# would only write the results to the GPRs whose predicate bit is set. The use of the P0 or P1 without precharging the sequencer with a PRED instruction is undefined.

{Issue: do we have to have a NOP between PRED and the first instruction that uses a predicate?}

~~7.4~~6.4 HW Detection of PV,PS

Because of the control program, the compiler cannot detect statically dependant instructions. In the case of non-masked writes and subsequent reads the sequencer will insert uses of PV,PS as needed. This will be done by comparing the read address and the write address of consecutive instructions. For masked writes, the sequencer will insert NOPs wherever there is a dependant read/write.

The sequencer will also have to insert NOPs between PRED_SET and MOVA instructions and their uses.

~~7.5~~6.5 Register file indexing

Because we can have loops in fetch clause, we need to be able to index into the register file in order to retrieve the data created in a fetch clause loop and use it into an ALU clause. The instruction will include the base address for register indexing and the instruction will contain these controls:

Bit7	Bit 6	
0	0	'absolute register'
0	1	'relative register'
1	0	'previous vector'
1	1	'previous scalar'

In the case of an absolute register we just take the address as is. In the case of a relative register read we take the base address and we add to it the loop_index and this becomes our new address that we give to the shader pipe.

The sequencer is going to keep a loop index computed as such:

Index = Loop_iterator*Loop_step + Loop_start.

We loop until loop_iterator = loop_count. Loop_step is a signed value [-128…127]. The computed index value is a 10 bit counter that is also signed. Its real range is [-256,256]. The tenth bit is only there so that we can provide an out of range value to the "indexing logic" so that it knows when the provided index is out of range and thus can make the necessary arrangements.

~~7.6~~6.6 Predicated Instruction support for Texture clauses

For texture clauses, we support the following optimization: we keep 1 bit (thus 4 bits for the four predicate vectors) per predicate vector in the reservation stations. A value of 1 means that one ore more elements in the vector have a value of one (thus we have to do the texture fetches for the whole vector). A value of 0 means that no elements in the vector have his predicate bit set and we can thus skip over the texture fetch. **We have to make sure the invalid pixels aren't considered with this optimization.**

~~7.7~~6.7 Debugging the Shaders

In order to be able to debug the pixel/vertex shaders efficiently, we provide 2 methods.

~~7.7.1~~6.7.1 *Method 1: Debugging registers*

Current plans are to expose 2 debugging, or error notification, registers:
1. address register where the first error occurred
2. count of the number of errors

The sequencer will detect the following groups of errors:
- count overflow
- constant indexing overflow
- register indexing overflow

Compiler recognizable errors:
- jump errors
  relative jump address > size of the control flow program
- call stack
  call with stack full
  return with stack empty

A jump error will always cause the program to break. In this case, a break means that a clause will halt execution, but allowing further clauses to be executed.

With all the other errors, program can continue to run, potentially to worst-case limits. The program will only break if the DB_PROB_BREAK register is set.

If indexing outside of the constant or the register range, causing an overflow error, the hardware is specified to return the value with an index of 0. This could be exploited to generate error tokens, by reserving and initializing the 0th register (or constant) for errors.

{ISSUE : Interrupt to the driver or not?}

~~7.7.2~~6.7.2 *Method 2: Exporting the values in the GPRs (12)*

The sequencer will have a count register and an address register for this mode and 3 bits per clause specifying the execution mode for each clause. The modes can be :

1) Normal
2) Debug Kill
3) Debug Addr + Count

Under the normal mode execution follows the normal course. Under the kill mode, all control flow instructions are executed but all normal shader instructions of the clause are replaced by NOPs. Only debug_export instructions of clause 7 will be executed under the debug kill setting. Under the other mode, normal execution is done until we reach an address specified by the address register and instruction count (useful for loops) specified by the count register. After we have hit the instruction n times (n=count) we switch the clause to the kill mode.

Under the debug mode (debug kill OR debug Addr + count), it is assumed that clause 7 is always exporting 12 debug vectors and that all other exports to the SX block (position, color, z, ect) will been turned off (changed into NOPs) by the sequencer (even if they occur before the address stated by the ADDR debug register).

~~8.~~7. Pixel Kill Mask

A vector of 64 bits is kept by the sequencer per group of pixels/vertices. Its purpose is to optimize the texture fetch requests and allow the shader pipe to kill pixels using the following instructions:

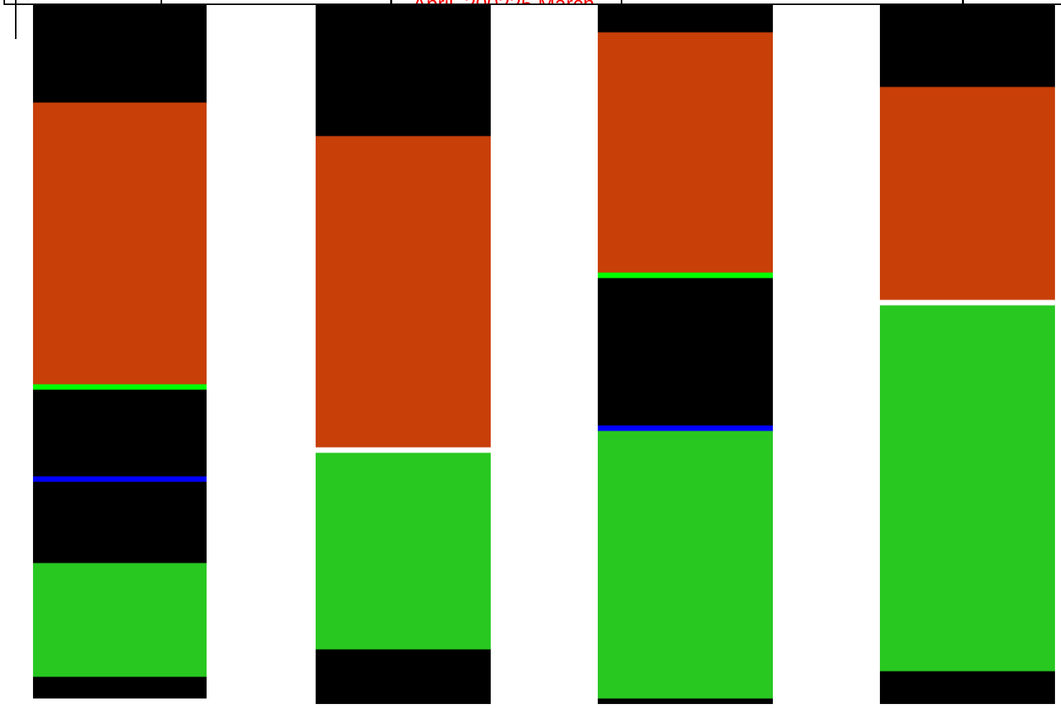
MASK_SETE
MASK_SETNE
MASK_SETGT
MASK_SETGTE

~~9.~~8. Multipass vertex shaders (HOS)

Multipass vertex shaders are able to export from the 6 last clauses but to memory ONLY.

~~10.~~9. Register file allocation

The register file allocation for vertices and pixels can either be static or dynamic. In both cases, the register file in managed using two round robins (one for pixels and one for vertices). In the dynamic case the boundary between pixels and vertices is allowed to move, in the static case it is fixed to 128-VERTEX_REG_SIZE for vertices and PIXEL_REG_SIZE for pixels.



Above is an example of how the algorithm works. Vertices come in from top to bottom; pixels come in from bottom to top. Vertices are in orange and pixels in green. The blue line is the tail of the vertices and the green line is the tail of the pixels. Thus anything between the two lines is shared. When pixels meets vertices the line turns white and the boundary is static until both vertices and pixels share the same "unallocated bubble". Then the boundary is allowed to move again. The numbering of the GPRs starts from the bottom of the picture at index 0 and goes up to the top at index 127.

~~11.~~10. Fetch Arbitration

The fetch arbitration logic chooses one of the 8 potentially pending fetch clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute. Once chosen, the clause state machine will send one 2x2 fetch per clock (or 4 fetches in one clock every 4 clocks) until all the fetch instructions of the clause are sent. This means that there cannot be any dependencies between two fetches of the same clause.

The arbitrator will not wait for the fetches to return prior to selecting another clause for execution. The fetch pipe will be able to handle up to X(?) in flight fetches and thus there can be a fair number of active clauses waiting for their fetch return data.

~~12.~~11. ALU Arbitration

ALU arbitration proceeds in almost the same way than fetch arbitration. The ALU arbitration logic chooses one of the 8 potentially pending ALU clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute. There are two ALU arbiters, one for the even clocks and one for the odd clocks. For example, here is the sequencing of two interleaved ALU clauses (E and O stands for Even and Odd sets of 4 clocks):

Einst0 Oinst0 Einst1 Oinst1 Einst2 Oinst2 Einst0 Oinst3 Einst1 Oinst4 Einst2 Oinst0…
Proceeding this way hides the latency of 8 clocks of the ALUs. Also note that the interleaving also occurs across clause boundaries.

~~13.~~12. Handling Stalls

When the output file is full, the sequencer prevents the ALU arbitration logic from selecting the last clause (this way nothing can exit the shader pipe until there is place in the output file. If the packet is a vertex packet and the position buffer is full (POS_FULL) then the sequencer also prevents a thread from entering the exporting clause (3?). The sequencer will set the OUT_FILE_FULL signal n clocks before the output file is actually full and thus the ALU arbiter will be able read this signal and act accordingly by not preventing exporting clauses to proceed.

~~14.~~13. Content of the reservation station FIFOs

The reservation FIFOs contain the state of the vector of pixels and vertices. We have two sets of those: one for pixels, and one for vertices. They contain 3 bits of Render State 7 bits for the base address of the GPRs, some bits for LOD correction and coverage mask information in order to fetch fetch for only valid pixels, the quad address.

~~15.~~14. The Output File

The output file is where pixels are put before they go to the RBs. The write BW to this store is 256 bits/clock. Just before this output file are staging registers with write BW 512 bits/clock and read BW 256 bits/clock. The staging registers are 4x128 (and there are 16 of those on the whole chip).

~~16.~~15. IJ Format

The IJ information sent by the PA is of this format on a per quad basis:

We have a vector of IJ's (one IJ per pixel at the centroid of the fragment or at the center of the pixel depending on the mode bit). The interpolation is done at a different precision across the 2x2. The upper left pixel's parameters are always interpolated at full 20x24 mantissa precision. Then the result of the interpolation along with the difference in IJ in reduced precision is used to interpolate the parameter for the other three pixels of the 2x2. Here is how we do it:

Assuming P0 is the interpolated parameter at Pixel 0 having the barycentric coordinates I(0), J(0) and so on for P1,P2 and P3. Also assuming that A is the parameter value at V0 (interpolated with I), B is the parameter value at V1 (interpolated with J) and C is the parameter value at V2 (interpolated with (1-I-J).

$$\begin{aligned}
\Delta 01I &= I(1) - I(0)\\
\Delta 01J &= J(1) - J(0)\\
\Delta 02I &= I(2) - I(0)\\
\Delta 02J &= J(2) - J(0)\\
\Delta 03I &= I(3) - I(0)\\
\Delta 03J &= J(3) - J(0)
\end{aligned}$$

P0	P1
P2	P3

$$\begin{aligned}
P0 &= C + I(0) * (A - C) + J(0) * (B - C)\\
P1 &= P0 + \Delta 01I * (A - C) + \Delta 01J * (B - C)\\
P2 &= P0 + \Delta 02I * (A - C) + \Delta 02J * (B - C)\\
P3 &= P0 + \Delta 03I * (A - C) + \Delta 03J * (B - C)
\end{aligned}$$

P0 is computed at 20x24 mantissa precision and P1 to P3 are computed at 8X24 mantissa precision. So far no visual degradation of the image was seen using this scheme.

Multiplies (Full Precision): 2
Multiplies (Reduced precision): 6
Subtracts 19x24 (Parameters): 2

Adds: 8

FORMAT OF P0's IJ : Mantissa 20 Exp 4 for I + Sign
Mantissa 20 Exp 4 for J + Sign

FORMAT of Deltas (x3):Mantissa 8 Exp 4 for I + Sign
Mantissa 8 Exp 4 for J + Sign

Total number of bits : 20*2 + 8*6 + 4*8 + 4*2 = 128

All numbers are kept using the un-normalized floating point convention: if exponent is different than 0 the number is normalized if not, then the number is un-normalized. The maximum range for the IJs (Full precision) is +/- 63 and the range for the Deltas is +/- 127.

~~16.1~~15.1 Interpolation of constant attributes

Because of the floating point imprecision, we need to take special provisions if all the interpolated terms are the same or if two of the barycentric coordinates are the same.

We start with the premise that if A = B and B = C and C = A, then P0,1,2,3 = A. Since one or more of the IJ terms may be zero, so we extend this to:

```
if (A=B and B=C and C=A)
   P0,1,2,3 = A;
else if ((I = 0) or (J = 0)) and
          ((J = 0) or (1-I-J = 0)) and
          ((1-J-I = 0) or (I = 0))) {
                if(I != 0) {
                    P0 = A;
                } else if(J != 0) {
                    P0 = B;
                } else {
                    P0 = C;
               }
            //rest of the quad interpolated normally
}
else
{
            normal interpolation
}
```

~~17.~~16. Staging Registers

In order for the reuse of the vertices to be 14, the sequencer will have to re-order the data sent IN ORDER by the VGT for it to be aligned with the parameter cache memory arrangement. Given the following group of vertices sent by the VGT:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 || 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 || 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 || 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63

The sequencer will re-arrange them in this fashion:

0 1 2 3 16 17 18 19 32 33 34 35 48 49 50 51 || 4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55 || 8 9 10 11 24 25 26 27 40 41 42 43 56 57 58 59 || 12 13 14 15 28 29 30 31 44 45 46 47 60 61 62 63

The || markers show the SP divisions. In the event a shader pipe is broken, the VGT will send padding to account for the missing pipe. For example, if SP1 is broken, vertices 4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55 will still be sent by the VGT to the SQ **BUT** will not be processed by the SP and thus should be considered invalid (by the SU and VGT).

The most straightforward, *non-compressed* interface method would be to convert, in the VGT, the data to 32-bit floating point prior to transmission to the VSISRs. In this scenario, the data would be transmitted to (and stored in) the VSISRs in full 32-bit floating point. This method requires three 24-bit fixed-to-float converters in the VGT. Unfortunately, it also requires and additional 3,072 bits of storage across the VSISRs. This interface is illustrated in Figure 11~~Figure 12Figure 12~~. The area of the fixed-to-float converters and the VSISRs for this method is roughly estimated as 0.759sqmm using the R300 process. The gate count estimate is shown in Figure 10~~Figure 11Figure 11~~.

Basis for 8-deep Latch Memory (from R300)			
8x24-bit	11631 μ^2	60.57813 μ^2 per bit	
Area of 96x8-deep Latch Memory	46524 μ^2		
Area of 24-bit Fix-to-float Converter	4712 μ^2 per converter		

Method 1	Block	Quantity	Area
	F2F	3	14136
	8x96 Latch	16	744384
			758520 μ^2

Figure 10~~11~~:Area Estimate for VGT to Shader Interface

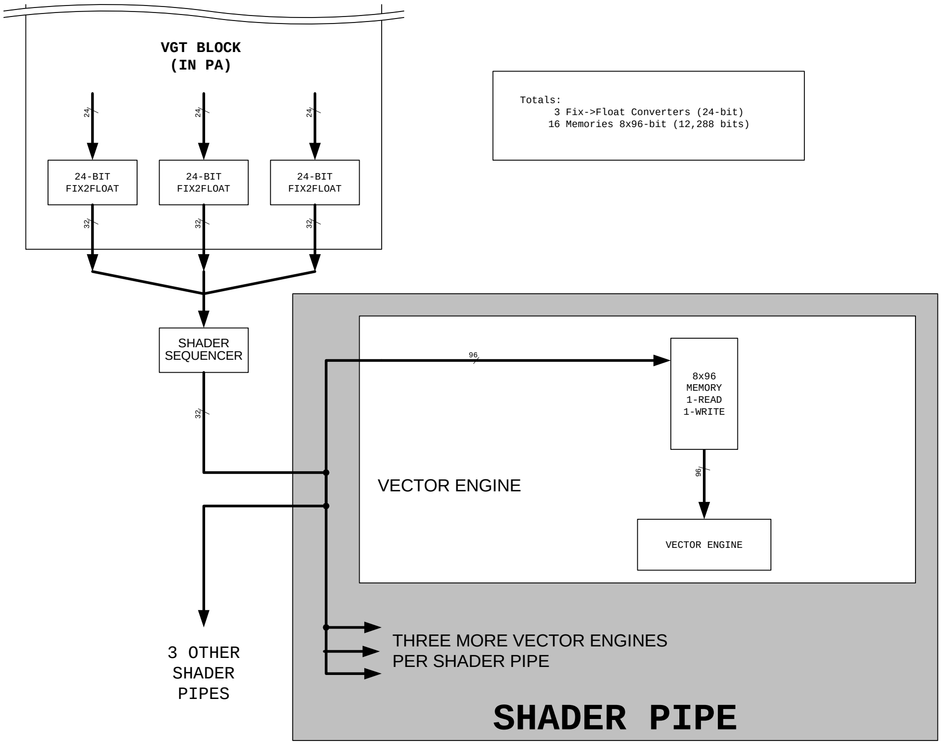


Figure 11~~12~~:VGT to Shader Interface

~~18.~~17. The parameter cache

The parameter cache is where the vertex shaders export their data. It consists of 16 128x128 memories (1R/1W). The reuse engine will make it so that all vertexes of a given primitive will hit different memories. The allocation method for these memories is a simple round robin. The parameter cache pointers are mapped in the following way: 4MSBs are the memory number and the 7 LSBs are the address within this memory.

MEMORY NUMBER 4 bits	ADDRESS 7 bits

The PA generates the parameter cache addresses as the positions come from the SQ. All it needs to do is keep a Current_Location pointer (7 bits only) and as the positions comes increment the memory number. When the memory number field wraps around, the PA increments the Current_Location by VS_EXPORT_COUNT_7 (a snooped register from the SQ). As an example, say the memories are all empty to begin with and the vertex shader is exporting 8 parameters per vertex (VS_EXPORT_COUNT_7 = 8). The first position received is going to have the PC address 00000000000 the second one 00010000000, third one 00100000000 and so on up to 11110000000. Then the next position received (the 17th) is going to have the address 00000001000, the 18th 00010001000, the 19th 00100001000 and so on. The Current_location is NEVER reset BUT on chip resets. The only thing to be careful about is that if the SX doesn't send you a full group of positions (<64) then you need to fill the address space so that the next group starts correctly aligned (for example if you receive only 33 positions then you need to add 2*VS_EXPORT_COUNT_7to Current_Location and reset the memory count to 0 before the next vector begins).

~~19.~~18. Vertex position exporting

On clause 3 the vertex shader can export to the PA both the vertex position and the point sprite. It can also do so at clause 7 if not done at clause 3. The storage needed to perform the position export is at least 64x128 memories for the position and 64x32 memories for the sprite size. It is going to be taken in the pixel output fifo from the SX blocks. The clause where the position export occurs is specified by the EXPORT_LATE register. If turned on, it means that the export is going to occur at ALU clause 7 if unset position export occurs at clause 3.

~~20.~~19. Exporting Arbitration

Here are the rules for co-issuing exporting ALU clauses.

1) Position exports and position exports cannot be co-issued.

All other types of exports can be co-issued as long as there is place in the receiving buffer.

~~21.~~20. Exporting Rules

~~21.1~~20.1 Parameter caches exports

We support masking and out of order exports to the parameter caches. So one can export multiple times to the same PC line using different masks.

~~21.2~~20.2 Memory exports

Memory exports don't support masking. However, you can export out of order to memory locations.

~~21.3~~20.3 Position exports

Position exports have to be done IN ORDER and don't support masking.

~~22.~~21. Export Types

The export type (or the location where the data should be put) is specified using the destination address field in the ALU instruction. Here is a list of all possible export modes:

~~22.1~~21.1 Vertex Shading

0:15	- 16 parameter cache
16:~~31~~31	- Empty (Reserved?)
32	- Export Address
~~32~~33:~~43~~ 40	- ~~12~~ 8 vertex exports to the frame buffer and index
~~44~~41:47	- Empty
48:~~59~~55	- ~~12~~ 8 debug export (interpret as normal vertex export)
60	- export addressing mode
61	- Empty
62	- position
63	- sprite size export that goes with position export (point_h,point_w,edgeflag,misc)

~~22.2~~21.2 Pixel Shading

0	- Color for buffer 0 (primary)
1	- Color for buffer 1
2	- Color for buffer 2
3	- Color for buffer 3
4:7	- Empty

8	- Buffer 0 Color/Fog (primary)
9	- Buffer 1 Color/Fog
10	- Buffer 2 Color/Fog
11	- Buffer 3 Color/Fog
12:15	- Empty
16:~~31~~31	- Empty (Reserved?)
32	- Export Address
~~32~~33:~~43~~40	- ~~12~~ 8 exports for multipass pixel shaders.
~~44~~41:47	- Empty
48:~~59~~55	- ~~12~~ 8 debug exports (interpret as normal pixel export)
60	- export addressing mode
61:62	- Empty
63	- Z for primary buffer (Z exported to 'alpha' component)

~~23.~~22. Special Interpolation modes

~~23.1~~22.1 Real time commands

We are unable to use the parameter memory since there is no way for a command stream to write into it. Instead we need to add three 16x128 memories (one for each of three vertices x 16 interpolants). These will be mapped onto the register bus and written by type 0 packets, and output to the the parameter busses (the sequencer and/or PA need to be able to address the realtime parameter memory as well as the regular parameter store. For higher performance we should be able able to view them as two banks of 16 and do double buffering allowing one to be loaded, while the other is rasterized with. Most overlay shaders will need 2 or 4 scalar coordinates, one option might be to restrict the memory to 16x64 or 32x64 allowing only two interpolated scalars per cycle, the only problem I see with this is, if we view support for 16 vector-4 interpolants important (true only if we map Microsoft's high priority stream to the realtime stream), then the PA/sequencer need to support a realtime-specific mode where we need to address 32 vectors of parameters instead of 16. This mode is triggered by the primitive type: REAL TIME. The actual memories are in the in the SX blocks. The parameter data memories are hooked on the RBBM bus and are loaded by the CP using register mapped memory.

~~23.2~~22.2 Sprites/ XY screen coordinates/ FB information

When working with sprites, one may want to overwrite the parameter 0 with SC generated data. Also, XY screen coordinates may be needed in the shader program. This functionality is controlled by the gen_I0 register (in SQ) in conjunction with the SND_XY register (in SC). Also it is possible to send the faceness information (for OGL front/back special operations) to the shader using the same control register. Here is a list of all the modes and how they interact together:

Gen_st is a bit taken from the interface between the SC and the SQ. This is the MSB of the primitive type. If the bit is set, it means we are dealing with Point AA, Line AA or sprite and in this case the vertex values are going to generated between 0 and 1.

Param_Gen_I0 disable, snd_xy disable, no gen_st – I0 = No modification
Param_Gen_I0 disable, snd_xy disable, gen_st – I0 = No modification
Param_Gen_I0 disable, snd_xy enable, no gen_st – I0 = No modification
Param_Gen_I0 disable, snd_xy enable, gen_st – I0 = No modification
Param_Gen_I0 enable, snd_xy disable, no gen_st – I0 = garbage, garbage, garbage, faceness
Param_Gen_I0 enable, snd_xy disable, gen_st – I0 = garbage, garbage, s, t
Param_Gen_I0 enable, snd_xy enable, no gen_st – I0 = screen x, screen y, garbage, faceness
Param_Gen_I0 enable, snd_xy enable, gen_st – I0 = screen x, screen y, s, t

~~23.3~~22.3 Auto generated counters

In the cases we are dealing with multipass shaders, the sequencer is going to generate a vector count to be able to both use this count to write the 1st pass data to memory and then use the count to retrieve the data on the 2nd pass. The count is always generated in the same way but it is passed to the shader in a slightly different way depending on the shader type (pixel or vertex). This is toggled on and off using the GEN_INDEX register. The sequencer is going to keep two counters, one for pixels and one for vertices. Every time a full vector of vertices or pixels is written to the

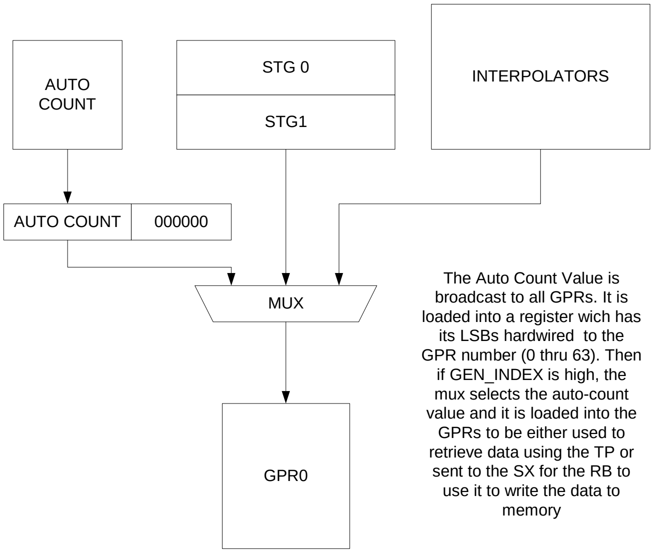
GPRs the counter is incremented. Every time a state change is detected, the corresponding counter is reset. While there is only one count broadcast to the GPRs, the LSB are hardwired to specific values making the index different for all elements in the vector.

~~23.3.1~~22.3.1 *Vertex shaders*

In the case of vertex shaders, if GEN_INDEX is set, the data will be put into the x field of the third register (it means that the compiler must allocate 3 GPRs in all multipass vertex shader modes).

~~23.3.2~~22.3.2 *Pixel shaders*

In the case of pixel shaders, if GEN_INDEX is set and Param_Gen_I0 is enabled, the data will be put in the x field of the 2nd register (R1.x), else if GEN_INDEX is set the data will be put into the x field of the 1st register (R0.x).



The Auto Count Value is broadcast to all GPRs. It is loaded into a register wich has its LSBs hardwired to the GPR number (0 thru 63). Then if GEN_INDEX is high, the mux selects the auto-count value and it is loaded into the GPRs to be either used to retrieve data using the TP or sent to the SX for the RB to use it to write the data to memory

Figure 12~~13~~: GPR input mux Control

~~24.~~23. State management

Every clock, the sequencer will report to the CP the oldest states still in the pipe. These are the states of the programs as they enter the last ALU clause.

~~24.1~~23.1 Parameter cache synchronization

In order for the sequencer not to begin a group of pixels before the associated group of vertices has finished, the sequencer will keep a 6 bit count per state (for a total of 8 counters). These counters are initialized to 0 and every time a vertex shader exports its data TO THE PARAMETER CACHE, the corresponding pointer is incremented. When the SC sends a new vector of pixels with the SC_SQ_new_vector bit asserted, the sequencer will first check if the count is greater than 0 before accepting the transmission (it will in fact accept the transmission but then lower its ready to receive). Then the sequencer waits for the count to go to one and decrements it. The sequencer can then issue the group of pixels to the interpolators. Every time the state changes, the new state counter is initialized to 0.

~~25.~~24. XY Address imports

The SC will be able to send the XY addresses to the GPRs. It does so by interleaving the writes of the IJs (to the IJ buffer) with XY writes (to the XY buffer). Then when writing the data to the GPRs, the sequencer is going to interpolate the IJ data or pass the XY data thru a Fix→float converter and expander and write the converted values to the GPRs. The Xys are currently SCREEN SPACE COORDINATES. The values in the XY buffers will wrap. See section 22.2 for details on how to control the interpolation in this mode.

~~25.1~~24.1 Vertex indexes imports

In order to import vertex indexes, we have 16 8x96 staging registers. These are loaded one line at a time by the VGT block (96 bits). They are loaded in floating point format and can be transferred in 4 or 8 clocks to the GPRs.

~~26.~~25. Registers

~~26.1~~25.1 Control

REG_DYNAMIC	Dynamic allocation (pixel/vertex) of the register file on or off.
REG_SIZE_PIX	Size of the register file's pixel portion (minimal size when dynamic allocation turned on)
REG_SIZE_VTX	Size of the register file's vertex portion (minimal size when dynamic allocation turned on)
ARBITRATION_POLICY	policy of the arbitration between vertexes and pixels
~~INST_STORE_ALLOC~~	~~interleaved, separate~~
INST_BASE_VTX	start point for the vertex instruction store (RT always ends at vertex_base and Begins at 0)
INST_BASE_PIX	start point for the pixel shader instruction store
ONE_THREAD	debug state register. Only allows one program at a time into the GPRs
ONE_ALU	debug state register. Only allows one ALU program at a time to be executed (instead of 2)
INSTRUCTION	This is where the CP puts the base address of the instruction writes and type (auto-incremented on reads/writes) Register mapped
CONSTANTS	512*4 ALU constants + 32*6 Texture state 32 bits registers (logically mapped)
CONSTANTS_RT	256*4 ALU constants + 32*6 texture states? (physically mapped)
CONSTANT_EO_RT	This is the size of the space reserved for real time in the constant store (from 0 to CONSTANT_EO_RT). The re-mapping table operates on the rest of the memory
TSTATE_EO_RT	This is the size of the space reserved for real time in the fetch state store (from 0 to TSTATE_EO_RT). The re-mapping table operates on the rest of the memory
EXPORT_LATE	Controls whether or not we are exporting position from clause 3. If set, position exports occur at clause 7.

~~26.2~~25.2 Context

VS_FETCH_{0…7}	eight 8 bit pointers to the location where each clauses control program is located
VS_ALU_{0…7}	eight 8 bit pointers to the location where each clauses control program is located
PS_FETCH_{0…7}	eight 8 bit pointers to the location where each clauses control program is located
PS_ALU_{0…7}	eight 8 bit pointers to the location where each clauses control program is located
PS_BASE	base pointer for the pixel shader in the instruction store
VS_BASE	base pointer for the vertex shader in the instruction store
VS_CF_SIZE	size of the vertex shader (# of instructions in control program/2)
PS_CF_SIZE	size of the pixel shader (# of instructions in control program/2)
PS_SIZE	size of the pixel shader (cntl+instructions)
VS_SIZE	size of the vertex shader (cntl+instructions)
PS_NUM_REG	number of GPRs to allocate for pixel shader programs
VS_NUM_REG	number of GPRs to allocate for vertex shader programs
PARAM_SHADE	One 16 bit register specifying which parameters are to be gouraud shaded (0 = flat, 1 = gouraud)
PROVO_VERT	0 : vertex 0, 1: vertex 1, 2: vertex 2, 3: Last vertex of the primitive
PARAM_WRAP	64 bits: for which parameters (and channels (xyzw)) do we do the cyl wrapping (0=linear, 1=cylindrical).
PS_EXPORT_MODE	0xxxx : Normal mode 1xxxx : Multipass mode

	If normal, bbbz where bbb is how many colors (0-4) and z is export z or not
	If multipass 1-12 exports for color.
VS_EXPORT_MODE	0: position (1 vector), 1: position (2 vectors), 3:multipass
VS_EXPORT _COUNT_{0…6}	Six 4 bit counters representing the # of interpolated parameters exported in clause 7 (located in VS_EXPORT_COUNT_6) OR # of exported vectors to memory per clause in multipass mode (per clause)
PARAM_GEN_I0	Do we overwrite or not the parameter 0 with XY data and generated T and S values
GEN_INDEX	Auto generates an address from 0 to XX. Puts the results into R0-1 for pixel shaders and R2 for vertex shaders
CONST_BASE_VTX (9 bits)	Logical Base address for the constants of the Vertex shader
CONST_BASE_PIX (9 bits)	Logical Base address for the constants of the Pixel shader
CONST_SIZE_PIX (8 bits)	Size of the logical constant store for pixel shaders
CONST_SIZE_VTX (8 bits)	Size of the logical constant store for vertex shaders
INST_PRED_OPTIMIZE	Turns on the predicate bit optimization (if of, conditional_execute_predicates is always executed).
CF_BOOLEANS	256 boolean bits
CF_LOOP_COUNT	32x8 bit counters (number of times we traverse the loop)
CF_LOOP_START	32x8 bit counters (init value used in index computation)
CF_LOOP_STEP	32x8 bit counters (step value used in index computation)

~~27.~~26. DEBUG Registers

~~27.1~~26.1 Context

DB_PROB_ADDR	instruction address where the first problem occurred
DB_PROB_COUNT	number of problems encountered during the execution of the program
DB_PROB_BREAK	break the clause if an error is found.
DB_INST_COUNT	instruction counter for debug method 2
DB_BREAK_ADDR	break address for method number 2
DB_CLAUSE _MODE_ALU_{0…7}	clause mode for debug method 2 (0: normal, 1: addr, 2: kill)
DB_CLAUSE _MODE_FETCH_{0…7}	clause mode for debug method 2 (0: normal, 1: addr, 2: kill)

~~27.2~~26.2 Control

DB_ALUCST_MEMSIZE	Size of the physical ALU constant memory
DB_TSTATE_MEMSIZE	Size of the physical texture state memory

~~28.~~27. Interfaces

~~28.1~~27.1 External Interfaces

Whenever an x is used, it means that the bus is broadcast to all units of the same name. For example, if a bus is named SQ→SPx it means that SQ is going to broadcast the same information to all SP instances.

~~28.2~~27.2 SC to SP Interfaces

~~28.2.1~~27.2.1 *SC_SP#*

There is one of these interfaces at front of each of the SP (buffer to stage pixel interpolators). This interface transmits the I,J data for pixel interpolation. For the entire system, two quads per clock are transferred to the 4 SPs, so each of these 4 interfaces transmits one half of a quad per clock. The interface below describes a half of a quad worth of data.

The actual data which is transferred per quad is
Ref Pix I => S4.20 Floating Point I value
Ref Pix J => S4.20 Floating Point J value
Delta Pix I (x3) => S4.8 Floating Point Delta I value
Delta Pix J (x3) => S4.8 Floating Point Delta J value
This equates to a total of 128 bits which transferred over 2 clocks
and therefor needs an interface 64 bits wide

Additionally, X,Y data (12-bit unsigned fixed) is conditionally sent across this data bus over the same wires in an additional clock. The X,Y data is sent on the lower 24 bits of the data bus with faceness in the msb.
Transfers across these interfaces are synchronized with the SC_SQ IJ Control Bus transfers.

The data transfer across each of these busses is controlled by a IJ_BUF_INUSE_COUNT in the SC. Each time the SC has sent a pixel vector's worth of data to the SPs, he will increment the IJ_BUF_INUSE_COUNT count. Prior to sending the next pixel vectors data, he will check to make sure the count is less than MAX_BUFER_MINUS_2, if not the SC will stall until the SQ returns a pipelined pulse to decrement the count when he has scheduled a buffer free. Note: We could/may optimize for the case of only sending only IJ to use all the buffers to pre-load more. Currently it is planned for the SP to hold 2 double buffers of I,J data and two buffers of X,Y data, so if either X,Y or Centers and Centroids are on, then the SC can send two Buffers.

In at least the initial version, the SC shall send 16 quads per pixel vector even if the vector is not full. This will increment buffer write address pointers correctly all the time. (We may revisit this for both the SX,SP,SQ and add a EndOfVector signal on all interfaces to quit early. We opted for the simple mode first with a belief that only the end of packet and multiple new vector signals should cause a partial vector and that this would not really be significant performance hit.)

Name	Bits	Description
SC_SP#_data	64	IJ information sent over 2 clocks (or X,Y in 24 LSBs with faceness in upper bit) **Type 0 or 1**, First clock I, second clk J Field ULC URC LLC LRC Bits [63:39] [38:26] [25:13] [12:0] Format SE4M20 SE4M8 SE4M8 SE4M8 **Type 2** Field Face X Y Bits [63] [23:12] [11:0] Format Bit Unsigned Unsigned
SC_SP#_valid	1	Valid
SC_SP#_last_quad_data	1	This bit will be set on the last transfer of data per quad.
SC_SP#_type	2	0 -> Indicates centroids 1 -> Indicates centers 2 -> Indicates X,Y Data and faceness on data bus The SC shall look at state data to determine how many types to send for the interpolation process.

The # is included for clarity in the spec and will be replaced with a prefix of u#_ in the verilog module statement for the SC and the SP block will have neither because the instantiation will insert the prefix.

~~28.2.2~~27.2.2 *SC_SQ*

This is the control information sent to the sequencer in order to synchronize and control the interpolation and/or loading data into the GPRs needed to execute a shader program on the sent pixels. This data will be sent over two clocks per transfer with 1 to 16 transfers. Therefore the bus (approx 92 bits) could be folded in half to approx ~~46~~ 47 bits.

Name	Bits	Description
SC_SQ_data	46	Control Data sent to the SQ 1 clk transfers Event – valid data consist of event_id and

		state_id. Instruct SQ to post an event vector to send state id and event_id through request fifo and onto the reservation stations making sure state id and/or event_id gets back to the CP. Events only follow end of packets so no pixel vectors will be in progress. Empty Quad Mask – Transfer Control data consisting of pc_dealloc or new_vector. Receipt of this is to transfer pc_dealloc or new_vector without any valid quad data. New vector will always be posted to request fifo and pc_dealloc will be attached to any pixel vector outstanding or posted in request fifo if no valid quad outstanding. 2 clk transfers Quad Data Valid – Sending quad data with or without new_vector or pc_dealloc. New vector will be posted to request fifo with or without a pixel vector and pc_dealloc will be posted with a pixel vector unless none is in progress. In this case the pc_dealloc will be posted in the request queue. Filler quads will be transferred with The Quad mask set but the pixel corresponding pixel mask set to zero.
SC_SQ_valid	1	SC sending valid data, 2nd clk could be all zeroes

SC_SQ_data – first clock and second clock transfers are shown in the table below.

Name	**BitField**	**Bits**	**Description**
1st Clock Transfer			
SC_SQ_event	0	1	This transfer is a 1 clock event vector Force quad_mask = new_vector=pc_dealloc=0
SC_SQ_event_id	[2:1]	2	This field identifies the event 0 => denotes an End Of State Event 1 => TBD
SC_SQ_pc_dealloc	[5:3]~~3~~	3~~1~~	Deallocation token for the Parameter Cache
SC_SQ_new_vector	6~~4~~	1	The SQ must wait for Vertex shader done count > 0 and after dispatching the Pixel Vector the SQ will decrement the count.
SC_SQ_quad_mask	[10~~8~~:7~~5~~]	4	Quad Write mask left to right SP0 => SP3
SC_SQ_end_of_prim	11~~9~~	1	End Of the primitive
SC_SQ_state_id	[14~~2~~:12~~0~~]	3	State/constant pointer (6*3+3)
SC_SQ_pix_mask	[30~~28~~:15~~3~~]	16	Valid bits for all pixels SP0=>SP3 (UL,UR,LL,LR)
SC_SQ_prim_type	[33~~1~~:31~~29~~]	3	Stippled line and Real time command need to load tex cords from alternate buffer 000: Normal ~~100~~010: Realtime 101: Line AA

			110: Point AA (Sprite)
SC_SQ_provok_vtx~~SC_SQ_pc_ptr0~~	[35:34]~~[42:32]~~	2~~11~~	Provoking vertex for flat shading~~Parameter Cache pointer for vertex 0~~
SC_SQ_pc_ptr0	[46:36]	11	Parameter Cache pointer for vertex 0
2nd Clock Transfer			
SC_SQ_pc_ptr1	[10:0]	11	Parameter Cache pointer for vertex 1
SC_SQ_pc_ptr2	[21:11]	11	Parameter Cache pointer for vertex 2
SC_SQ_lod_correct	[45:22]	24	LOD correction per quad (6 bits per quad)

Name	Bits	Description
SQ_SC_free_buff	1	Pipelined bit that instructs SC to decrement count of buffers in use.
SQ_SC_dec_cntr_cnt	1	Pipelined bit that instructs SC to decrement count of new vector and/or event sent to prevent SC from overflowing SQ interpolator/Reservation request fifo.

The scan converter will submit a partial vector whenever:

1.) He gets a primitive marked with an end of packet signal.
2.) A current pixel vector is being assembled with at least one or more valid quads and the vector has been marked for deallocate when a primitive marked new_vector arrives. The Scan Converter will submit a partial vector (up to 16quads with zero pixel mask to fill out the vector) prior to submitting the new_vector marker\primitive.

(This will prevent a hang which can be demonstrated when all primitives in a packet three vectors are culled except for a one quad primitive that gets marked pc_dealloc (vertices maximum size). In this case two new_vectors are submitted and processed, but then one valid quad with the pc_dealloc creates a vector and then the new would wait for another vertex vector to be processed, but the one being waited for could never export until the pc_dealloc signal made it through and thus the hang.)

~~28.2.3~~27.2.3 *SQ to SX: Interpolator bus*

Name	Direction	Bits	Description
SQ_SXx_interp_flat_vtx	SQ→SPx	2	Provoking vertex for flat shading
SQ_SXx_interp_flat_gouraud	SQ→SPx	1	Flat or gouraud shading
SQ_SXx_interp_cyl_wrap	SQ→SPx	4	Wich channel needs to be cylindrical wrapped
SQ_SXx_pc_ptr0~~1~~	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_pc_ptr1~~2~~	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_pc_ptr2~~3~~	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_rt_sel	SQ→SXx	1	Selects between RT and Normal data
SQ_SXx_pc_wr_en	SQ→SXx	1	Write enable for the PC memories
SQ_SXx_pc_wr_addr	SQ→SXx	7	Write address for the PCs
SQ_SXx_pc_channel_mask	SQ→SXx	4	Channel mask

~~28.2.4~~27.2.4 *SQ to SP: Staging Register Data*

This is a broadcast bus that sends the VSISR information to the staging registers of the shader pipes.

Name	Direction	Bits	Description
SQ_SPx_vsr_data	SQ→SPx	96	Pointers of indexes or HOS surface information
SQ_SPx_vsr_double	SQ→SPx	1	0: Normal 96 bits per vert 1: double 192 bits per vert
SQ_SP0_vsr_valid	SQ→SP0	1	Data is valid
SQ_SP1_vsr_valid	SQ→SP1	1	Data is valid
SQ_SP2_vsr_valid	SQ→SP2	1	Data is valid
SQ_SP3_vsr_valid	SQ→SP3	1	Data is valid
SQ_SPx_vsr_read	SQ→SPx	1	Increment the read pointers

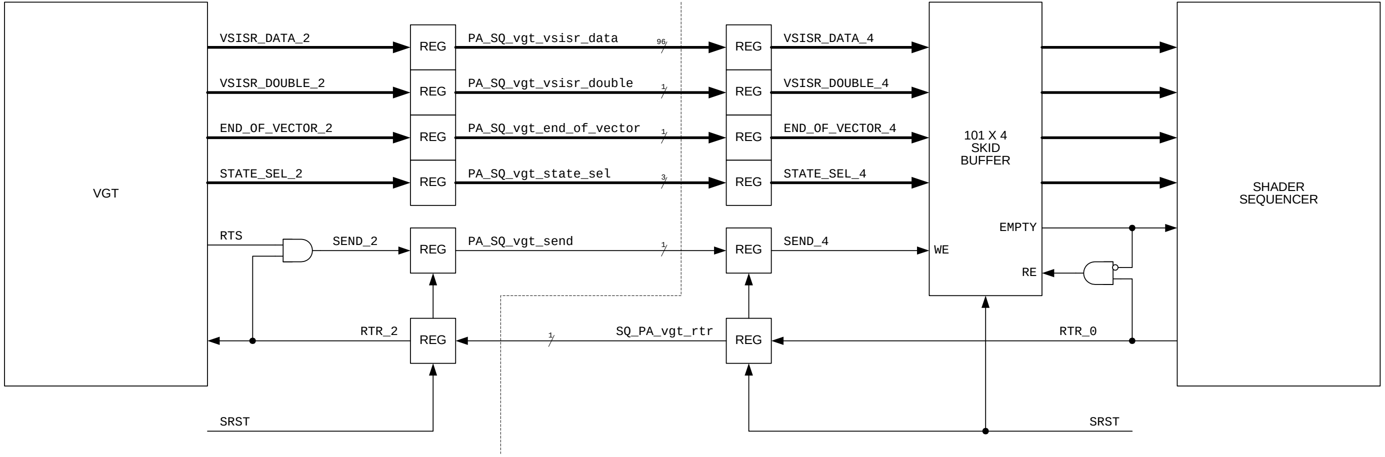
~~28.2.5~~27.2.5 *VGT to SQ : Vertex interface*

~~28.2.5.1~~27.2.5.1 Interface Signal Table

The area difference between the two methods is not sufficient to warrant complicating the interface or the state requirements of the VSISRs. **Therefore, the POR for this interface is that the VGT will transmit the data to the VSISRs (via the Shader Sequencer) in full, 32-bit floating-point format.** The VGT can transmit up to six 32-bit floating-point values to each VSISR where four or more values require two transmission clocks. The data bus is 96 bits wide.

Name	Bits	Description
VGT_SQ_vsisr_data	96	Pointers of indexes or HOS surface information
VGT_SQ_vsisr_double	1	0: Normal 96 bits per vert 1: double 192 bits per vert
VGT_SQ_end_of_vector	1	Indicates the last VSISR data set for the current process vector (for double vector data, "end_of_vector" is set on the ~~second~~ first vector)
VGT_SQ_indx_valid	1	Vsisr data is valid
VGT_SQ_state	3	Render State (6*3+3 for constants). This signal is guaranteed to be correct when "VGT_SQ_vgt_end_of_vector" is high.
VGT_SQ_send	1	Data on the VGT_SQ is valid receive (see write-up for standard R400 SEND/RTR interface handshaking)
SQ_VGT_rtr	1	Ready to receive (see write-up for standard R400 SEND/RTR interface handshaking)

~~28.2.5.2~~27.2.5.2 Interface Diagrams


VGT
VSISR_DATA_2
VSISR_DOUBLE_2
END_OF_VECTOR_2
STATE_SEL_2
RTS
SEND_2
RTR_2
SRST
REG
REG
REG
REG
REG
REG
PA_SQ_vgt_vsisr_data
96
PA_SQ_vgt_vsisr_double
1
PA_SQ_vgt_end_of_vector
1
PA_SQ_vgt_state_sel
3
PA_SQ_vgt_send
1
1
SQ_PA_vgt_rtr
REG
REG
REG
REG
REG
REG
VSISR_DATA_4
VSISR_DOUBLE_4
END_OF_VECTOR_4
STATE_SEL_4
SEND_4
101 X 4
SKID
BUFFER
EMPTY
WE
RE
RTR_0
SRST
SHADER
SEQUENCER

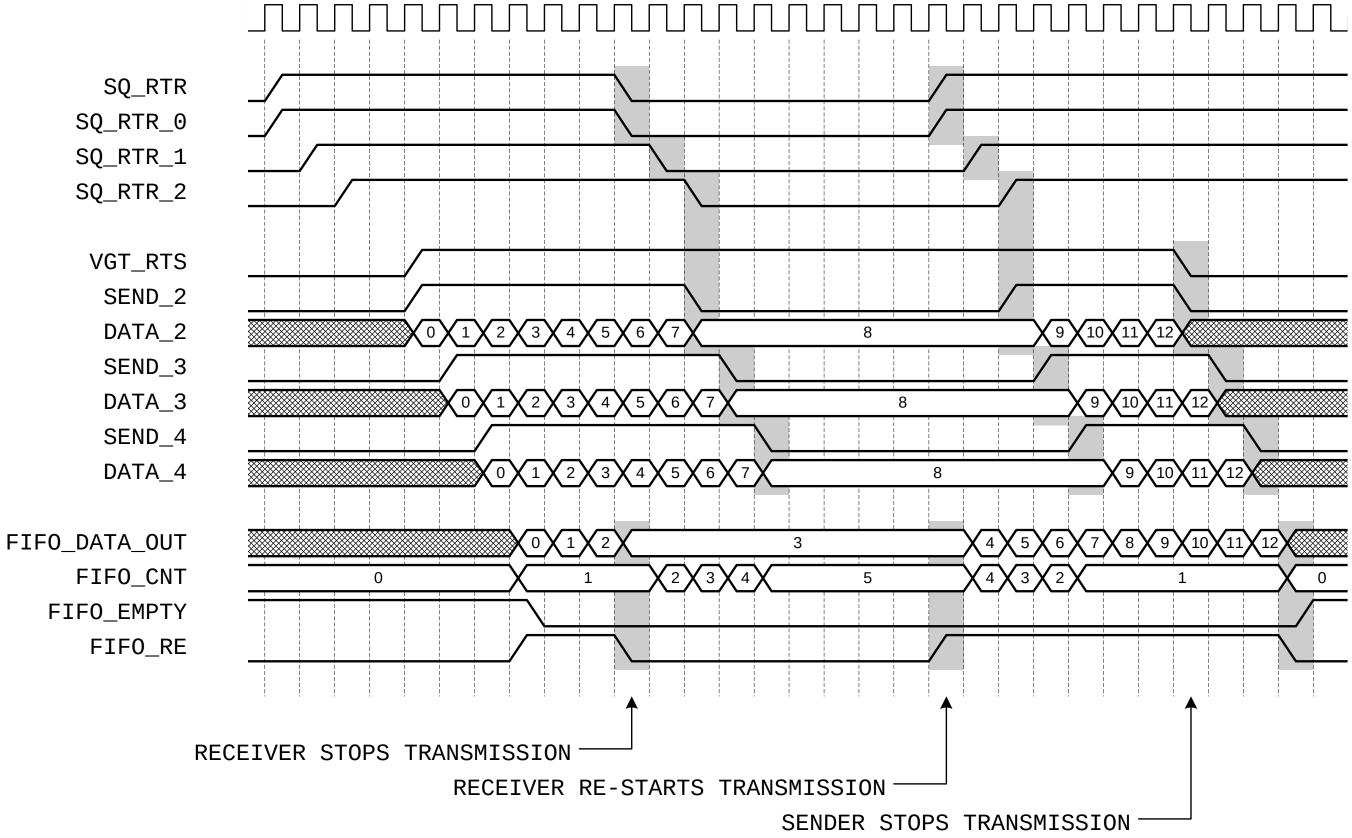


Figure 1. Detailed Logical Diagram for PA_SQ_vgt Interface.

~~27.2.6 SQ to CP: State report~~

Formatted: Bullets and Numbering

~~27.2.7~~27.2.6 SQ to SX: Control bus

Name	Direction	Bits	Description
SQ_SXx_exp_pix	SQ→SXx	1	1: Pixel 0: Vertex
SQ_SXx_exp_clause	SQ→SXx	3	Clause number, which is needed for vertex clauses
SQ_SXx_exp_state	SQ→SXx	3	State ID
SQ_SXx_exp_alu_id	SQ→SXx	1	ALU ID
SQ_SXx_exp_valid	SQ→SXx	1	Valid bit

These fields are sent every time the sequencer picks an exporting clause for execution.

~~27.2.8~~27.2.7 SX to SQ : Output file control

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SXx_SQ_exp_count_rdy	SXx→SQ	1	Raised by SX0 to indicate that the following two fields reflect the result of the most recent export
SXx_SQ_exp_pos~~ition~~_avail~~space~~	SXx→SQ	1	Specifies whether there is room for another position.
SXx_SQ_exp_bu~~ffer~~_avail~~space~~	SXx→SQ	7	Specifies the space available in the output buffers. 0: buffers are full 1: 2K-bits available (32-bits for each of the 64 pixels in a clause) ... 64: 128K-bits available (16 128-bit entries for each of 64 pixels) 65-127: RESERVED

~~27.2.9~~27.2.8 SQ to TP: Control bus

Formatted: Bullets and Numbering

Once every clock, the fetch unit sends to the sequencer on which clause it is now working and if the data in the GPRs is ready or not. This way the sequencer can update the fetch counters for the reservation station fifos. The sequencer also provides the instruction and constants for the fetch to execute and the address in the register file where to write the fetch return data.

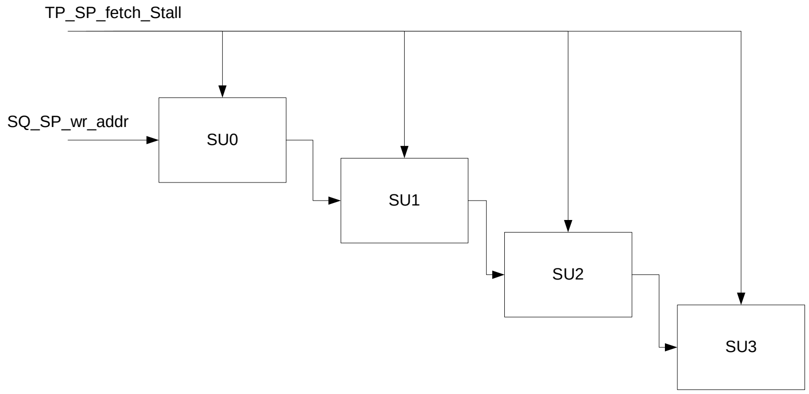
Name	Direction	Bits	Description
TPx_SQ_data_rdy	TPx→ SQ	1	Data ready
TPx_SQ_clause_num	TPx→ SQ	3	Clause number
TPx_SQ_type	TPx→ SQ	1	Type of data sent (0:PIXEL, 1:VERTEX)
SQ_TPx_send	SQ→TPx	1	Sending valid data
SQ_TPx_const	SQ→TPx	48	Fetch state sent over 4 clocks (192 bits total)
SQ_TPx_instr	SQ→TPx	24	Fetch instruction sent over 4 clocks
SQ_TPx_end_of_clause	SQ→TPx	1	Last instruction of the clause
SQ_TPx_Type	SQ→TPx	1	Type of data sent (0:PIXEL, 1:VERTEX)
SQ_TPx_gpr_phase	SQ→TPx	2	Write phase signal
SQ_TP0_lod_correct	SQ→TP0	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP0_pix_mask	SQ→TP0	4	Pixel mask 1 bit per pixel
SQ_TP1_lod_correct	SQ→TP1	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP1_pix_mask	SQ→TP1	4	Pixel mask 1 bit per pixel
SQ_TP2_lod_correct	SQ→TP2	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP2_pix_mask	SQ→TP2	4	Pixel mask 1 bit per pixel
SQ_TP3_lod_correct	SQ→TP3	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP3_pix_mask	SQ→TP3	4	Pixel mask 1 bit per pixel
SQ_TPx_clause_num	SQ→TPx	3	Clause number

SQ_TPx_write_gpr_index	SQ->TPx	7	Index into Register file for write of returned Fetch Data

~~27.2.10~~27.2.9 TP to SQ: Texture stall

Formatted: Bullets and Numbering

The TP sends this signal to the SQ and the SPs when its input buffer is full. ~~The SQ is going to send it to the SP X clocks after reception (maximum of 3 clocks of pipeline delay).~~



Name	Direction	Bits	Description
TP_SQ_fetch_stall	TP→ SQ	1	Do not send more texture request if asserted

~~27.2.11~~27.2.10 SQ to SP: Texture stall

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SPx_fetch_stall	SQ→SPx	1	Do not send more texture request if asserted

~~27.2.12~~27.2.11 SQ to SP: GPR and auto counter

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SPx_gpr_wr_addr	SQ→SPx	7	Write address
SQ_SPx_gpr_rd_addr	SQ→SPx	7	Read address
SQ_SPx_gpr_rd_en	SQ→SPx	1	Read Enable
SQ_SPx_gpr_wr_en	SQ→SPx	1	Write Enable for the GPRs
SQ_SPx_gpr_phase~~_mux~~	SQ→SPx	2	The phase mux (arbitrates between inputs, ALU SRC reads and writes)
SQ_SPx_channel_mask	SQ→SPx	4	The channel mask
SQ_SPx_gpr_input_~~mux~~sel	SQ→SPx	2	When the phase mux selects the inputs this tells from which source to read from: Interpolated data, VTX0, VTX1, autogen counter.
SQ_SPx_auto_count	SQ→SPx	12?	Auto count generated by the SQ, common for all shader pipes

~~27.2.13~~27.2.12 SQ to SPx: Instructions

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SPx_instr_start	SQ→SPx	1	Instruction start
SQ_SP_instr	SQ→SPx	21	Transferred over 4 cycles 0: SRC A Select 2:0 SRC A Argument Modifier 3:3 SRC A swizzle 11:4 VectorDst 17:12 Unused 20:18 -- - 1: SRC B Select 2:0 SRC B Argument Modifier 3:3 SRC B swizzle 11:4 ScalarDst 17:12 Unused 20:18 -- - 2: SRC C Select 2:0 SRC C Argument Modifier 3:3 SRC C swizzle 11:4 Unused 20:12 -- - 3: Vector Opcode 4:0 Scalar Opcode 10:5 Vector Clamp 11:11 Scalar Clamp 12:12 Vector Write Mask 16:13 Scalar Write Mask 20:17
SQ_SPx_exp_alu_id	SQ→SPx	1	ALU ID
SQ_SPx_exporting	SQ→SPx	2	0: Not Exporting 1: Vector Exporting 2: Scalar Exporting
SQ_SPx_stall	SQ→SPx	1	Stall signal
SQ_SP0_~~exp_pvalid~~write_mask	SQ→SP0	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP1_ write_mask~~exp_pvalid~~	SQ→SP1	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP2_ write_mask~~exp_pvalid~~	SQ→SP2	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP3_ write_mask~~exp_pvalid~~	SQ→SP3	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock

~~27.2.14~~27.2.13 SP to SQ: Constant address load/ Predicate Set

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SP0_SQ_const_addr	SP0→SQ	36	Constant address load / predicate vector load (4 bits only) to the sequencer
SP0_SQ_valid	SP0→SQ	1	Data valid
SP1_SQ_const_addr	SP1→SQ	36	Constant address load / predicate vector load (4 bits only) to the sequencer

SP1_SQ_valid	SP1→SQ	1	Data valid
SP2_SQ_const_addr	SP2→SQ	36	Constant address load / predicate vector load (4 bits only) to the sequencer
SP2_SQ_valid	SP2→SQ	1	Data valid
SP3_SQ_const_addr	SP3→SQ	36	Constant address load / predicate vector load (4 bits only) to the sequencer
SP3_SQ_valid	SP3→SQ	1	Data valid

~~27.2.15~~27.2.14 SQ to SPx: constant broadcast

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SPx_const	SQ→SPx	128	Constant broadcast

~~27.2.16~~27.2.15 SP0 to SQ: Kill vector load

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SP0_SQ_kill_vect	SP0→SQ	4	Kill vector load
SP1_SQ_kill_vect	SP1→SQ	4	Kill vector load
SP2_SQ_kill_vect	SP2→SQ	4	Kill vector load
SP3_SQ_kill_vect	SP3→SQ	4	Kill vector load

~~27.2.17~~27.2.16 SQ to CP: RBBM bus

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_RBB_rs	SQ→CP	1	Read Strobe
SQ_RBB_rd	SQ→CP	32	Read Data
SQ_RBBM_nrtrtr	SQ→CP	1	Optional
SQ_RBBM_rtr	SQ→CP	1	Real-Time (Optional)

~~27.2.18~~27.2.17 CP to SQ: RBBM bus

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
rbbm_we	CP→SQ	1	Write Enable
rbbm_a	CP→SQ	15	Address -- Upper Extent is TBD (16:2)
rbbm_wd	CP→SQ	32	Data
rbbm_be	CP→SQ	4	Byte Enables
rbbm_re	CP→SQ	1	Read Enable
rbb_rs0	CP→SQ	1	Read Return Strobe 0
rbb_rs1	CP→SQ	1	Read Return Strobe 1
rbb_rd0	CP→SQ	32	Read Data 0
rbb_rd1	CP→SQ	32	Read Data 0
RBBM_SQ_soft_reset	CP→SQ	1	Soft Reset

27.2.18 SQ to CP: State report

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_CP_vs_event	SQ→CP	1	Vertex Shader Event
SQ_CP_vs_eventid	SQ→CP	2	Vertex Shader Event ID
SQ_CP_ps_event	SQ→CP	1	Pixel Shader Event
SQ_CP_ps_eventid	SQ→CP	2	Pixel Shader Event ID

eventid = 0 => *sEndOfState (i.e. VsEndOfState)
eventid = 1 => *sDone (i.e. VsDone)

So, the CP will assume the Vs is done with a state whenever it gets a pulse on the SQ_CP_vs_event and the SQ_CP_vs_eventid = 0.

Formatted: Bullets and Numbering

~~28.Examples of program executions~~

~~28.1.1 *Sequencer Control of a Vector of Vertices*~~

~~1.PA sends a vector of 64 vertices (actually vertex indices – 32 bits/index for 2048 bit total) to the RE's Vertex FIFO~~

- ~~state pointer as well as tag into position cache is sent along with vertices~~
- ~~space was allocated in the position cache for transformed position before the vector was sent~~
- ~~**also before the vector is sent to the RE, the CP has loaded the global instruction store with the vertex shader program (using the MH?)**~~
- ~~The vertex program is assumed to be loaded when we receive the vertex vector.~~
    - ~~the SEQ then accesses the IS base for this shader using the local state pointer (provided to all sequencers by the RBBM when the CP is done loading the program)~~

~~2.SEQ arbitrates between the Pixel FIFO and the Vertex FIFO – basically the Vertex FIFO always has priority~~

- ~~at this point the vector is removed from the Vertex FIFO~~
- ~~the arbiter is not going to select a vector to be transformed if the parameter cache is full unless the pipe as nothing else to do (ie no pixels are in the pixel fifo).~~

~~3.SEQ allocates space in the SP register file for index data plus GPRs used by the program~~

- ~~the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer that came down with the vertices~~
- ~~SEQ will not send vertex data until space in the register file has been allocated~~

~~4.SEQ sends the vector to the SP register file over the RE_SP interface (which has a bandwidth of 2048 bits/cycle)~~

- ~~the 64 vertex indices are sent to the 64 register files over 4 cycles~~
    - ~~RF0 of SU0, SU1, SU2, and SU3 is written the first cycle~~
    - ~~RF1 of SU0, SU1, SU2, and SU3 is written the second cycle~~
    - ~~RF2 of SU0, SU1, SU2, and SU3 is written the third cycle~~
    - ~~RF3 of SU0, SU1, SU2, and SU3 is written the fourth cycle~~
- ~~the index is written to the least significant 32 bits **(floating point format?) (what about compound indices)** of the 128-bit location within the register file (w); the remaining data bits are set to zero (x, y, z)~~

~~5.SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of fetch state machine 0, or TSM0 FIFO)~~

- ~~the control packet contains the state pointer, the tag to the position cache and a register file base pointer.~~

~~6.TSM0 accepts the control packet and fetches the instructions for fetch clause 0 from the global instruction store~~

- ~~TSM0 was first selected by the TSM arbiter before it could start~~

~~7.all instructions of fetch clause 0 are issued by TSM0~~

~~8.the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)~~

- ~~TSM0 does not wait for requests made to the Fetch Unit to complete; it passes the register file write index for the fetch data to the TU, which will write the data to the RF as it is received~~
- ~~once the TU has written all the data to the register files, it increments a counter that is associated with ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go ahead start to execute the ALU clause~~

~~9.ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store~~

~~10.all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of fetch state machine 1, or TSM1 FIFO)~~

~~11.the control packet continues to travel down the path of reservation stations until all clauses have been executed~~

- ~~position can be exported in ALU clause 3 (or 4?); the data (and the tag) is sent over a position bus (which is shared with all four shader pipes) back to the PA's position cache~~
- ~~A parameter cache pointer is also sent along with the position data. This tells to the PA where the data is going to be in the parameter cache.~~
    - ~~there is a position export FIFO in the SP that buffers position data before it gets sent back to the PA~~

    - ~~the ASM arbiter will prevent a packet from starting an exporting clause if the position export FIFO is full~~
  - ~~parameter data is exported in clause 7 (as well as position data if it was not exported earlier)~~
    - ~~parameter data is sent to the Parameter Cache over a dedicated bus~~
    - ~~the SEQ allocates storage in the Parameter Cache, and the SEQ deallocates that space when there is no longer a need for the parameters (it is told by the PA when using a token).~~
    - ~~the ASM arbiter will prevent a packet from starting on ASM7 if the parameter cache (or the position buffer if position is being exported) is full~~

~~12.after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program~~

~~28.1.2 *Sequencer Control of a Vector of Pixels*~~

~~1.As with vertex shader programs, pixel shaders are loaded into the global instruction store by the CP~~

- ~~At this point it is assumed that the pixel program is loaded into the instruction store and thus ready to be read.~~

~~2.the RE's Pixel FIFO is loaded with the barycentric coordinates for pixel quads by the detailed walker~~
- ~~the state pointer and the LOD correction bits are also placed in the Pixel FIF0~~
- ~~the Pixel FIFO is wide enough to source four quad's worth of barycentrics per cycle~~

~~3.SEQ arbitrates between Pixel FIFO and Vertex FIFO – when there are no vertices pending OR there is no space left in the register files for vertices, the Pixel FIFO is selected~~

~~4.SEQ allocates space in the SP register file for all the GPRs used by the program~~
- ~~the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer~~
- ~~SEQ will not allow interpolated data to be sent to the shader until space in the register file has been allocated~~

~~5.SEQ controls the transfer of interpolated data to the SP register file over the RE_SP interface (which has a bandwidth of 2048 bits/cycle). See interpolated data bus diagrams for details.~~

~~6.SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of fetch state machine 0, or TSM0 FIFO)~~
- ~~note that there is a separate set of reservation stations/arbiters/state machines for vertices and for pixels~~
- ~~the control packet contains the state pointer, the register file base pointer, and the LOD correction bits~~
- ~~all other information (such as quad address for example) travels in a separate FIFO~~

~~7.TSM0 accepts the control packet and fetches the instructions for fetch clause 0 from the global instruction store~~
- ~~TSM0 was first selected by the TSM arbiter before it could start~~

~~8.all instructions of fetch clause 0 are issued by TSM0~~

~~9.the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)~~
- ~~TSM0 does not wait for fetch requests made to the Fetch Unit to complete; it passes the register file write index for the fetch data to the TU, which will write the data to the RF as it is received~~
- ~~once the TU has written all the data for a particular clause to the register files, it increments a counter that is associated with the ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go ahead and pop the FIFO and start to execute the ALU clause~~

~~10.ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store~~

~~11.all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of fetch state machine 1, or TSM1 FIFO)~~

~~12.the control packet continues to travel down the path of reservation stations until all clauses have been executed~~
- ~~pixel data is exported in the last ALU clause (clause 7)~~
  - ~~it is sent to an output FIFO where it will be picked up by the render backend~~
  - ~~the ASM arbiter will prevent a packet from starting on ASM7 if the output FIFO is full~~

~~13.after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program~~

~~28.1.3 *Notes*~~

Formatted: Bullets and Numbering

~~14.The state machines and arbiters will operate ahead of time so that they will be able to immediately start the real threads or stall.~~

~~15.The register file base pointer for a vector needs to travel with the vector through the reservation stations, but the instruction store base pointer does not – this is because the RF pointer is different for all threads, but the IS pointer is only different for each state and thus can be accessed via the state pointer.~~

Formatted: Bullets and Numbering

~~29.~~28. Open issues

Need to do some testing on the size of the register file as well as on the register file allocation method (dynamic VS static).

Saving power?

~~Parameter caches in SX?~~

~~Using both IJ buffers for center + centroid interpolation?~~

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 2015~~19~~ April, 2002	GEN-CXXXXX-REVA	1 of 58

Author: Laurent Lefebvre

Issue To:	**Copy No:**

R400 Sequencer Specification

SQ

Version ~~1.11~~2.0

Overview: This is an architectural specification for the R400 Sequencer block (SEQ). It provides an overview of the required capabilities and expected uses of the block. It also describes the block interfaces, internal sub-blocks, and provides internal state diagrams.

AUTOMATICALLY UPDATED FIELDS:

Document Location: C:\perforce\r400\doc_lib\design\blocks\sq\R400_Sequencer.doc

Current Intranet Search Title: R400 Sequencer Specification

APPROVALS	
Name/Dept	Signature/Date

Remarks:

THIS DOCUMENT CONTAINS CONFIDENTIAL INFORMATION THAT COULD BE SUBSTANTIALLY DETRIMENTAL TO THE INTEREST OF ATI TECHNOLOGIES INC. THROUGH UNAUTHORIZED USE OR DISCLOSURE.

Table Of Contents

1. OVERVIEW 6
1.1 Top Level Block Diagram .... 9~~8~~
1.2 Data Flow graph (SP) .... 13~~10~~
1.3 Control Graph .... 14~~11~~
2. INTERPOLATED DATA BUS .... 14~~11~~
3. INSTRUCTION STORE .... 17~~14~~
4. SEQUENCER INSTRUCTIONS .... 17~~14~~
5. CONSTANT STORES .... 17~~14~~
5.1 Memory organizations .... 17~~14~~
5.2 Management of the Control Flow Constants .... 18~~15~~
5.3 Management of the re-mapping tables .... 18~~15~~
5.3.1 R400 Constant management .... 18~~15~~
5.3.2 Proposal for R400LE constant management .... 18~~15~~
5.3.3 Dirty bits .... 20~~17~~
5.3.4 Free List Block .... 20~~17~~
5.3.5 De-allocate Block .... 21~~18~~
5.3.6 Operation of Incremental model .... 21~~18~~
5.4 Constant Store Indexing .... 21~~18~~
5.5 Real Time Commands .... 22~~19~~
5.6 Constant Waterfalling .... 22~~19~~
6. LOOPING AND BRANCHES .... 23~~20~~
6.1 The controlling state. .... 23~~20~~
6.2 The Control Flow Program .... 23~~20~~
6.3 Data dependant predicate instructions .... 29~~22~~
6.4 HW Detection of PV,PS .... 29~~23~~
6.5 Register file indexing .... 29~~23~~
6.6 Predicated Instruction support for Texture clauses .... 30~~23~~
6.7 Debugging the Shaders .... 30~~23~~
6.7.1 Method 1: Debugging registers .... 30~~23~~
6.7.2 Method 2: Exporting the values in the GPRs (12) .... 30~~24~~
7. PIXEL KILL MASK .... 31~~24~~
8. MULTIPASS VERTEX SHADERS (HOS) .... 31~~24~~
9. REGISTER FILE ALLOCATION .... 31~~24~~
10. FETCH ARBITRATION .... 32~~26~~
11. ALU ARBITRATION .... 32~~26~~
12. HANDLING STALLS .... 33~~27~~
13. CONTENT OF THE RESERVATION STATION FIFOS .... 33~~27~~
14. THE OUTPUT FILE .... 33~~27~~
15. IJ FORMAT .... 33~~27~~
15.1 Interpolation of constant attributes .... 34~~28~~
16. STAGING REGISTERS .... 34~~28~~
17. THE PARAMETER CACHE .... 36~~30~~
18. VERTEX POSITION EXPORTING .... 37~~30~~

19. EXPORTING ARBITRATION .... **37~~30~~**
20. EXPORTING RULES .... **38~~30~~**
20.1 Parameter caches exports .... 38~~30~~
20.2 Memory exports .... 38~~30~~
20.3 Position exports .... 38~~30~~
21. EXPORT TYPES .... **38~~30~~**
21.1 Vertex Shading .... 38~~31~~
21.2 Pixel Shading .... 38~~31~~
22. SPECIAL INTERPOLATION MODES .... **39~~31~~**
22.1 Real time commands .... 39~~31~~
22.2 Sprites/ XY screen coordinates/ FB information .... 39~~31~~
22.3 Auto generated counters .... 39~~32~~
22.3.1 Vertex shaders .... 39~~32~~
22.3.2 Pixel shaders .... 39~~32~~
23. STATE MANAGEMENT .... **40~~33~~**
23.1 Parameter cache synchronization .... 40~~33~~
24. XY ADDRESS IMPORTS .... **40~~33~~**
24.1 Vertex indexes imports .... 40~~33~~
25. REGISTERS .... **41~~33~~**
25.1 Control .... 41~~33~~
25.2 Context .... 41~~33~~
26. DEBUG REGISTERS .... **42~~34~~**
26.1 Context .... 42~~34~~
26.2 Control .... 42~~34~~
27. INTERFACES .... **42~~35~~**
27.1 External Interfaces .... 42~~35~~
27.2 SC to SP Interfaces .... 42~~35~~
27.2.1 SC_SP# .... 42~~35~~
27.2.2 SC_SQ .... 43~~36~~
27.2.3 SQ to SX: Interpolator bus .... 45~~37~~
27.2.4 SQ to SP: Staging Register Data .... 45~~37~~
27.2.5 VGT to SQ : Vertex interface .... 45~~38~~
27.2.6 SQ to SX: Control bus .... 49~~41~~
27.2.7 SX to SQ : Output file control .... 49~~41~~
27.2.8 SQ to TP: Control bus .... 50~~41~~
27.2.9 TP to SQ: Texture stall .... 51~~42~~
27.2.10 SQ to SP: Texture stall .... 51~~42~~
27.2.11 SQ to SP: GPR and auto counter .... 51~~42~~
27.2.12 SQ to SPx: Instructions .... 52~~43~~
27.2.13 SP to SQ: Constant address load/ Predicate Set .... 52~~43~~
27.2.14 SQ to SPx: constant broadcast .... 53~~44~~
27.2.15 SP0 to SQ: Kill vector load .... 53~~44~~
27.2.16 SQ to CP: RBBM bus .... 53~~44~~

27.2.17 CP to SQ: RBBM bus .... 53~~44~~
27.2.18 SQ to CP: State report .... 53~~44~~
28. OPEN ISSUES .... 58~~44~~

Revision Changes:

Rev 0.1 (Laurent Lefebvre) Date: May 7, 2001	First draft.
Rev 0.2 (Laurent Lefebvre) Date : July 9, 2001	Changed the interfaces to reflect the changes in the SP. Added some details in the arbitration section.
Rev 0.3 (Laurent Lefebvre) Date : August 6, 2001	Reviewed the Sequencer spec after the meeting on August 3, 2001.
Rev 0.4 (Laurent Lefebvre) Date : August 24, 2001	Added the dynamic allocation method for register file and an example (written in part by Vic) of the flow of pixels/vertices in the sequencer.
Rev 0.5 (Laurent Lefebvre) Date : September 7, 2001	Added timing diagrams (Vic)
Rev 0.6 (Laurent Lefebvre) Date : September 24, 2001	Changed the spec to reflect the new R400 architecture. Added interfaces.
Rev 0.7 (Laurent Lefebvre) Date : October 5, 2001	Added constant store management, instruction store management, control flow management and data dependant predication.
Rev 0.8 (Laurent Lefebvre) Date : October 8, 2001	Changed the control flow method to be more flexible. Also updated the external interfaces.
Rev 0.9 (Laurent Lefebvre) Date : October 17, 2001	Incorporated changes made in the 10/18/01 control flow meeting. Added a NOP instruction, removed the conditional_execute_or_jump. Added debug registers.
Rev 1.0 (Laurent Lefebvre) Date : October 19, 2001	Refined interfaces to RB. Added state registers.
Rev 1.1 (Laurent Lefebvre) Date : October 26, 2001	Added SEQ→SP0 interfaces. Changed delta precision. Changed VGT→SP0 interface. Debug Methods added.
Rev 1.2 (Laurent Lefebvre) Date : November 16, 2001	Interfaces greatly refined. Cleaned up the spec.
Rev 1.3 (Laurent Lefebvre) Date : November 26, 2001	Added the different interpolation modes.
Rev 1.4 (Laurent Lefebvre) Date : December 6, 2001	Added the auto incrementing counters. Changed the VGT→SQ interface. Added content on constant management. Updated GPRs.
Rev 1.5 (Laurent Lefebvre) Date : December 11, 2001	Removed from the spec all interfaces that weren't directly tied to the SQ. Added explanations on constant management. Added PA→SQ synchronization fields and explanation.
Rev 1.6 (Laurent Lefebvre) Date : January 7, 2002	Added more details on the staging register. Added detail about the parameter caches. Changed the call instruction to a Conditionnal_call instruction. Added details on constant management and updated the diagram.
Rev 1.7 (Laurent Lefebvre) Date : February 4, 2002	Added Real Time parameter control in the SX interface. Updated the control flow section.
Rev 1.8 (Laurent Lefebvre) Date : March 4, 2002	New interfaces to the SX block. Added the end of clause modifier, removed the end of clause instructions.
Rev 1.9 (Laurent Lefebvre) Date : March 18, 2002	Rearangement of the CF instruction bits in order to ensure byte alignement.
Rev 1.10 (Laurent Lefebvre) Date : March 25, 2002	Updated the interfaces and added a section on exporting rules.
Rev 1.11 (Laurent Lefebvre) Date : April 19, 2002	Added CP state report interface. Last version of the spec with the old control flow scheme
Rev 2.0 (Laurent Lefebvre) Date : April 19, 2002	New control flow scheme

1. Overview

~~The sequencer is based on the R300 design. It~~The sequencer chooses two ALU ~~clauses~~ threads and a fetch ~~clause~~ hread to execute, and executes all of the instructions in a ~~clause~~ block before looking for a new clause of the same type. Two ALU ~~clauses~~ threads are executed interleaved to hide the ALU latency. ~~Each vector will have eight fetch and eight ALU clauses, but clauses do not need to contain instructions. A vector of pixels or vertices ping-pongs along the sequencer FIFO, bouncing from fetch reservation station to alu reservation station. A FIFO exists between each reservation stage, holding up vectors until the vector currently occupying a reservation station has left. A vector at a reservation station can be chosen to execute. The sequencer looks at all eight alu reservation stations to choose an alu clause to execute and all eight fetch stations to choose a fetch clause to execute.~~ The arbitrator will give priority to ~~clauses/reservation stations closer to the bottom of the pipeline~~older threads. ~~It will not execute an alu clause until the fetch fetches initiated by the previous fetch clause have completed.~~ There are two separate sets of reservation stations, one for pixel vectors and one for vertices vectors. This way a pixel can pass a vertex and a vertex can pass a pixel.

To support the shader pipe the sequencer also contains the shader instruction cache, constant store, control flow constants and texture state. The four shader pipes also execute the same instruction thus there is only one sequencer for the whole chip.

The sequencer first arbitrates between vectors of 64 vertices that arrive directly from primitive assembly and vectors of 16 quads (64 pixels) that are generated in the scan converter.

The vertex or pixel program specifies how many GPRs it needs to execute. The sequencer will not start the next vector until the needed space is available in the GPRs.

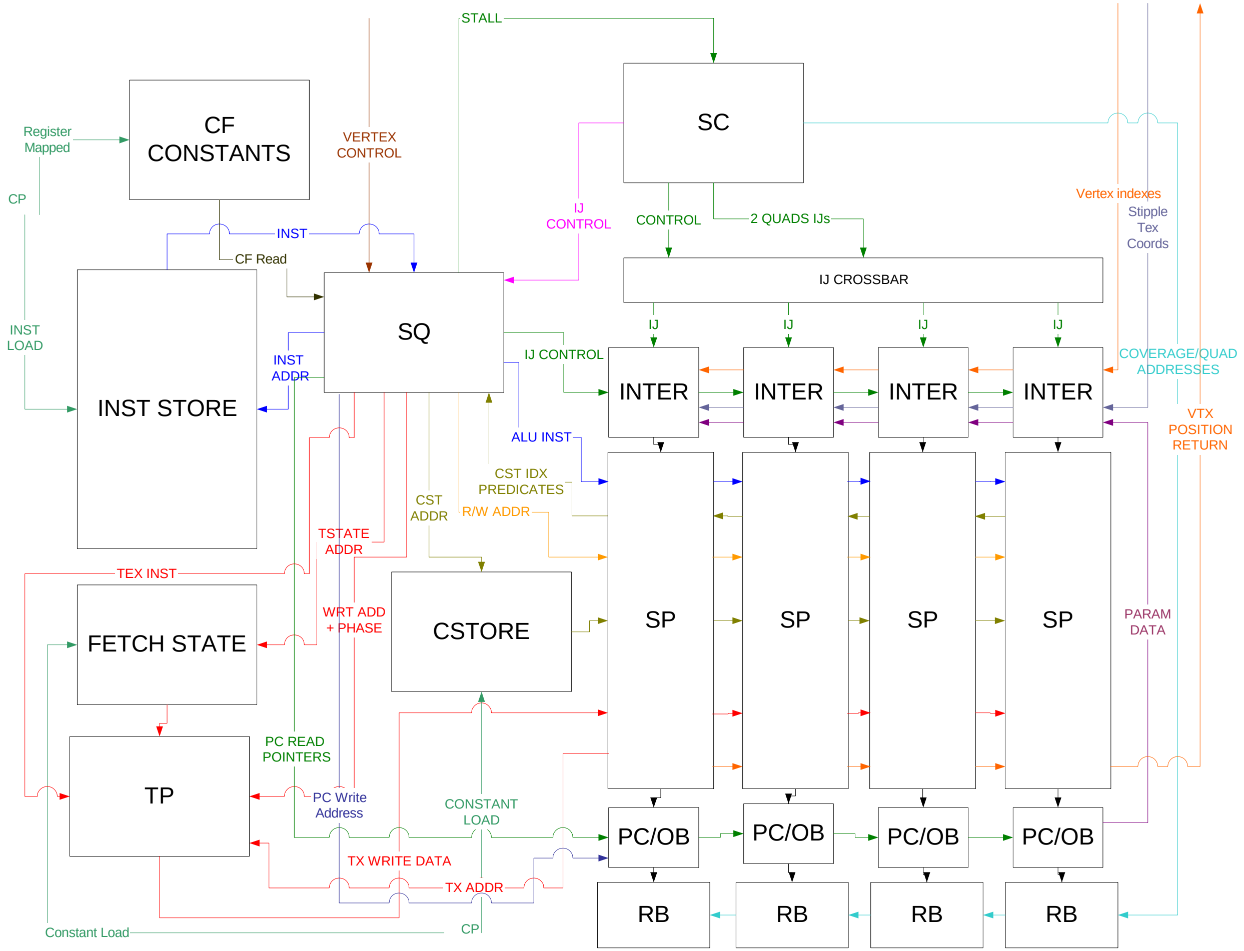
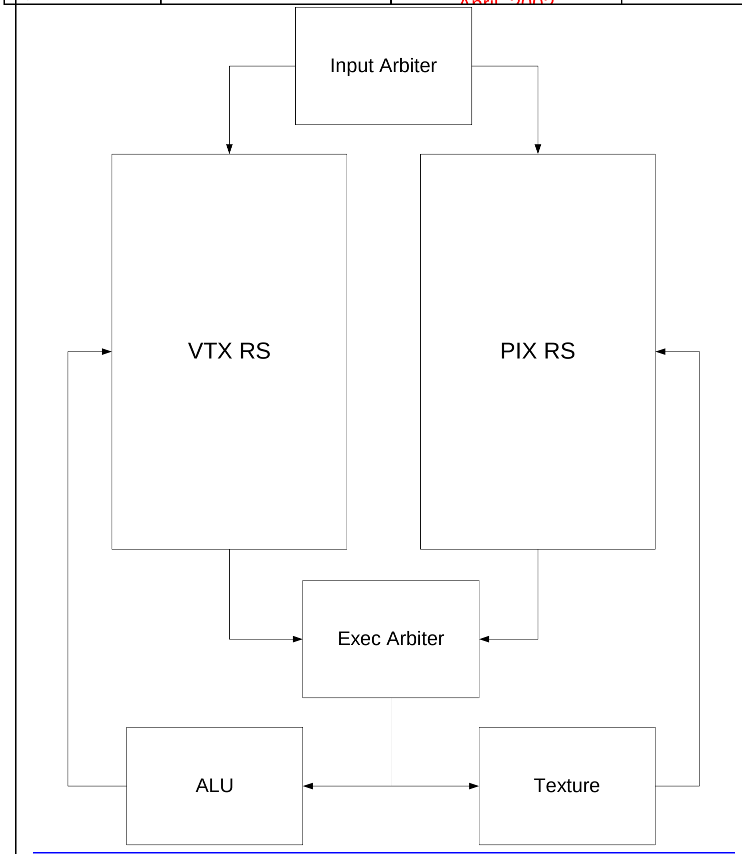
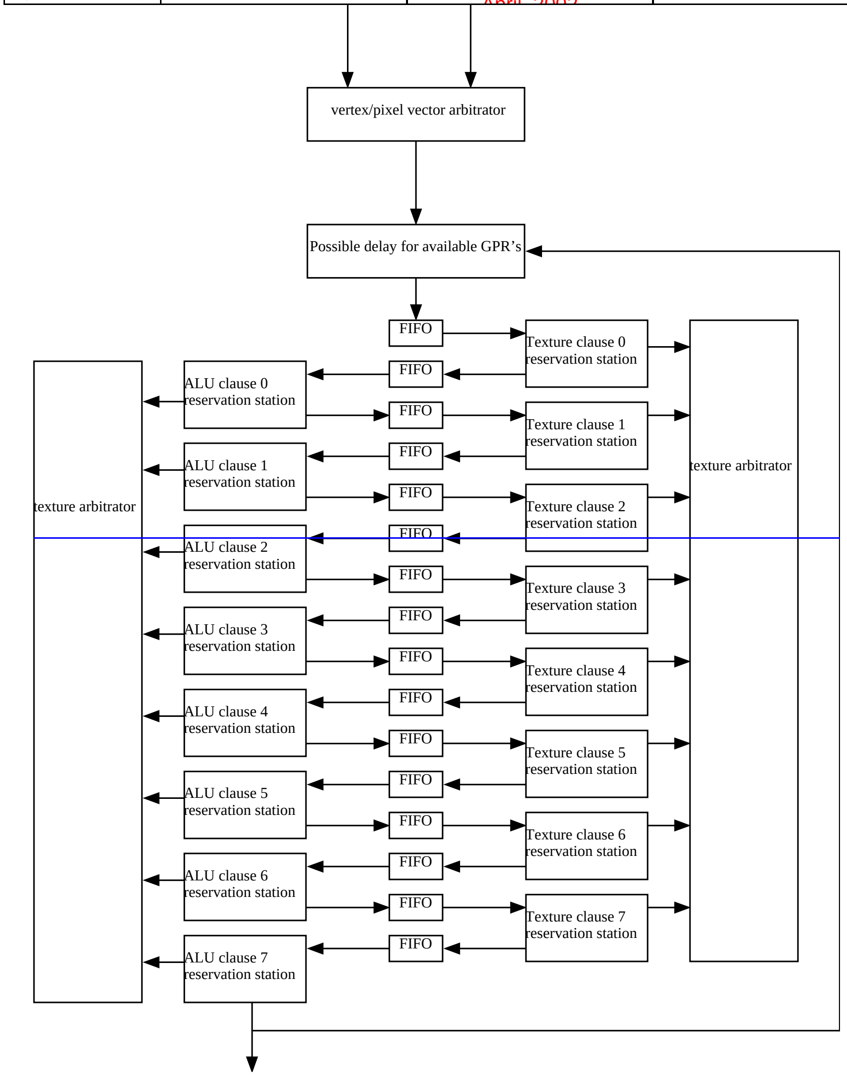


Figure 1: General Sequencer overview

1.1 Top Level Block Diagram


Input Arbiter
VTX RS
PIX RS
Exec Arbiter
ALU
Texture



vertex/pixel vector arbitrator
Possible delay for available GPR's
FIFO
Texture clause 0 reservation station
Texture clause 1 reservation station
Texture clause 2 reservation station
Texture clause 3 reservation station
Texture clause 4 reservation station
Texture clause 5 reservation station
Texture clause 6 reservation station
Texture clause 7 reservation station
ALU clause 0 reservation station
ALU clause 1 reservation station
ALU clause 2 reservation station
ALU clause 3 reservation station
ALU clause 4 reservation station
ALU clause 5 reservation station
ALU clause 6 reservation station
ALU clause 7 reservation station
texture arbitrator
texture arbitrator

Figure 2: Reservation stations and arbiters

~~There are two sets of the above figure, one for vertices and one for pixels.~~

~~Depending on the arbitration state, the sequencer will either choose a vertex or a pixel packet. The control packet consists of 3 bits of state, 7 bits for the base address of the Shader program and some information on the coverage to determine fetch LOD plus other various small state bits.~~

~~On receipt of a packet, the input state machine (not pictured but just before the first FIFO) allocated enough space in the GPRs to store the interpolated values and temporaries. Following this, the barycentric coordinates (and XY screen position if needed) are sent to the interpolator, which will use them to interpolate the parameters and place the results into the GPRs. Then, the input state machine stacks the packet in the first FIFO.~~

~~On receipt of a command, the level 0 fetch machine issues a fetch request to the TP and corresponding GPR address for the fetch address (ta). A small command (tcmd) is passed to the fetch system identifying the current level number (0) as well as the GPR write address for the fetch return data. One fetch request is sent every 4 clocks causing the texturing of sixteen 2x2s worth of data (or 64 vertices). Once all the requests are sent the packet is put in FIFO 1.~~

~~Upon receipt of the return data, the fetch unit writes the data to the register file using the write address that was provided by the level 0 fetch machine and sends the clause number (0) to the level 0 fetch state machine to signify that the write is done and thus the data is ready. Then, the level 0 fetch machine increments the counter of FIFO 1 to signify to the ALU 0 that the data is ready to be processed.~~

~~On receipt of a command, the level 0 ALU machine first decrements the input FIFO 1 counter and then issues a complete set of level 0 shader instructions. For each instruction, the ALU state machine generates 3 source addresses, one destination address and an instruction. Once the last instruction has been issued, the packet is put into FIFO 2.~~

~~There will always be two active ALU clauses at any given time (and two arbiters). One arbiter will arbitrate over the odd instructions (4 clocks cycles) and the other one will arbitrate over the even instructions (4 clocks cycles). The only constraints between the two arbiters is that they are not allowed to pick the same clause number as the other one is currently working on if the packet is not of the same type (render state).~~

~~If the packet is a vertex packet, upon reaching ALU clause 3, it can export the position if the position is ready. So the arbiter must prevent ALU clause 3 to be selected if the positional buffer is full (or can't be accessed). Along with the positional data, if needed the sprite size and/or edge flags can also be sent.~~

~~A special case is for multipass vertex shaders, which can export 12 parameters per last 6 clauses to the output buffer. If the output buffer is full or doesn't have enough space the sequencer will prevent such a vertex group to enter an exporting clause.~~

~~Multipass pixel shaders can export 12 parameters to memory from the last clause only (7).~~

~~All other clauses process in the same way until the packet finally reaches the last ALU machine (7).~~

~~Only one pair of interleaved ALU state machines may have access to the register file address bus or the instruction decode bus at one time. Similarly, only one fetch state machine may have access to the register file address bus at one time. Arbitration is performed by three arbiter blocks (two for the ALU state machines and one for the fetch state machines). The arbiters always favor the higher number state machines, preventing a bunch of half finished jobs from clogging up the register files.~~
Under this new scheme, the sequencer (SQ) will only use one global state management machine per vector type (pixel, vertex) that we call the reservation station (RS).

1.2 Data Flow graph (SP)

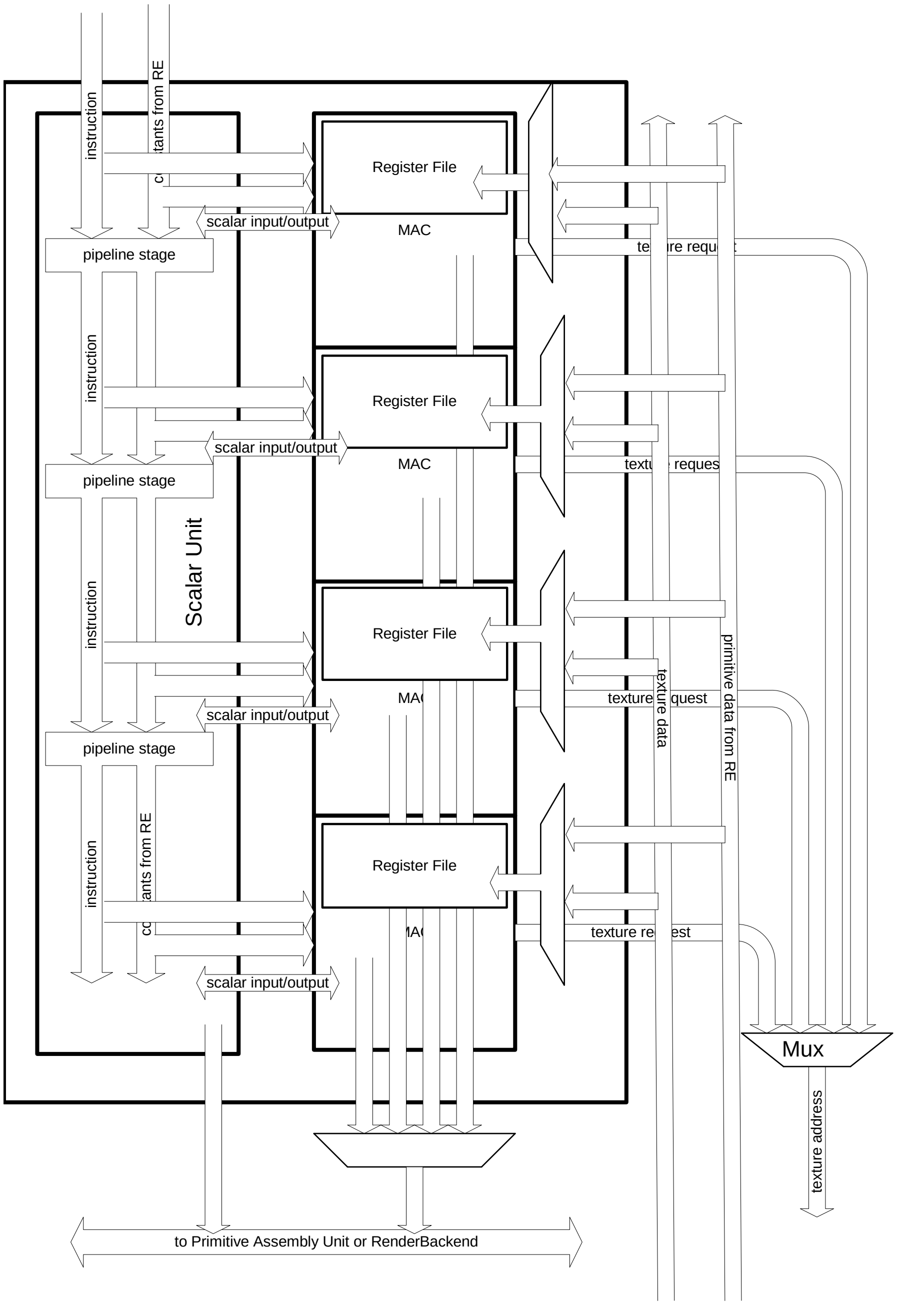


Figure 3: The shader Pipe

The gray area represents blocks that are replicated 4 times per shader pipe (16 times on the overall chip).

1.3 Control Graph

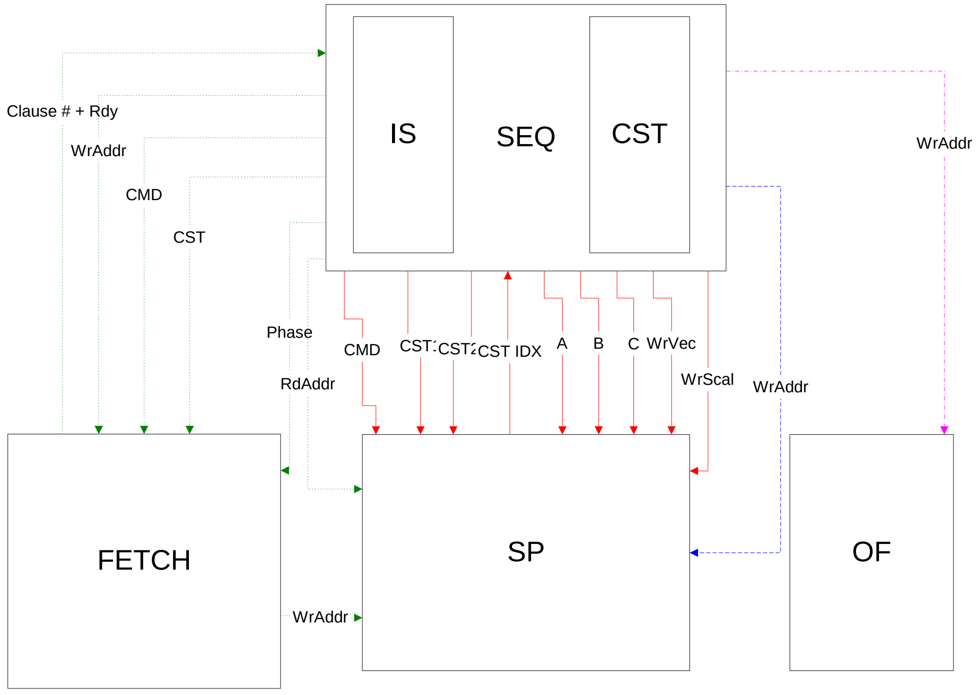


Figure 4: Sequencer Control interfaces

In green is represented the Fetch control interface, in red the ALU control interface, in blue the Interpolated/Vector control interface and in purple is the output file control interface.

2. Interpolated data bus

The interpolators contain an IJ buffer to pack the information as much as possible before writing it to the register file.

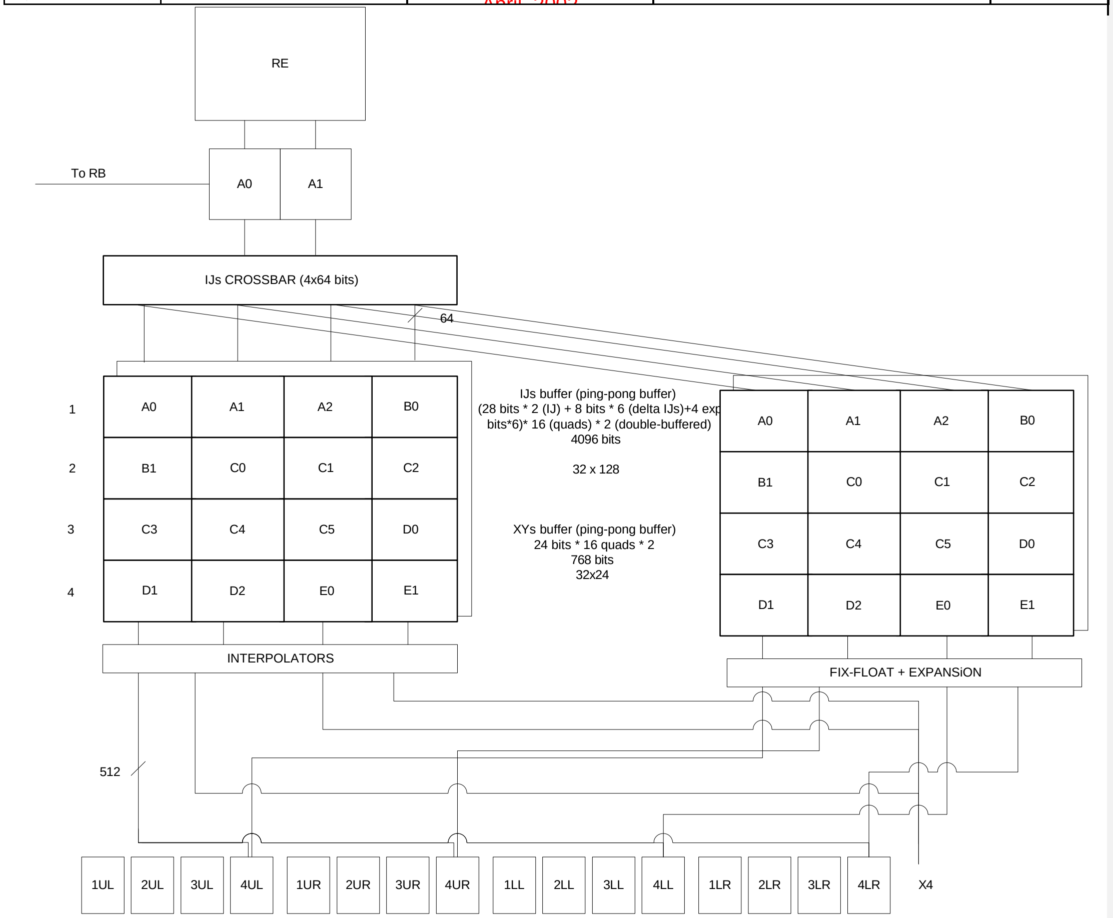
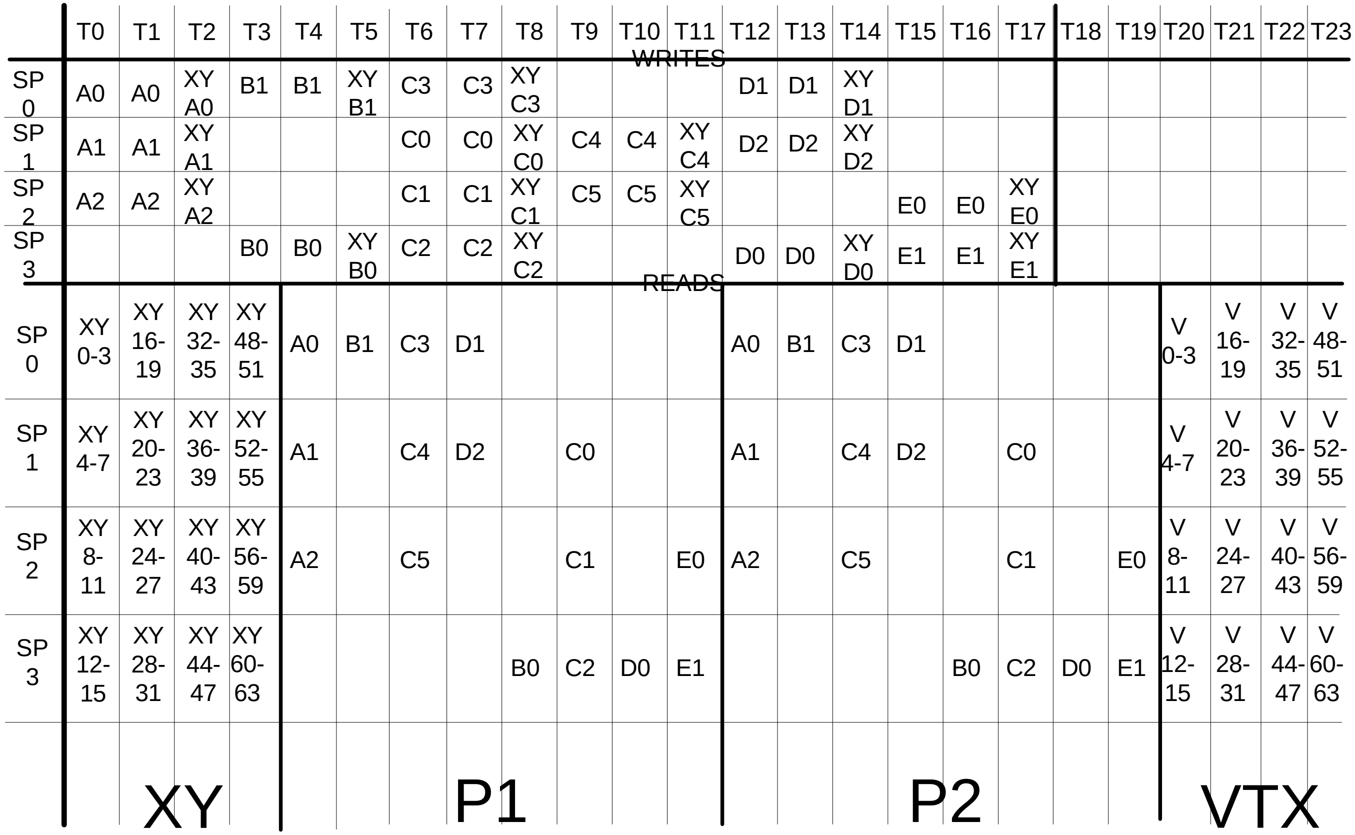


Figure 5: Interpolation buffers



	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21	T22	T23
											WRITES													
SP 0	A0	A0	XY A0	B1	B1	XY B1	C3	C3	XY C3				D1	D1	XY D1									
SP 1	A1	A1	XY A1				C0	C0	XY C0	C4	C4	XY C4	D2	D2	XY D2									
SP 2	A2	A2	XY A2				C1	C1	XY C1	C5	C5	XY C5				E0	E0	XY E0						
SP 3				B0	B0	XY B0	C2	C2	XY C2				D0	D0	XY D0	E1	E1	XY E1						
												READS												
SP 0	XY 0-3	XY 16-19	XY 32-35	XY 48-51	A0	B1	C3	D1					A0	B1	C3	D1					V 0-3	V 16-19	V 32-35	V 48-51
SP 1	XY 4-7	XY 20-23	XY 36-39	XY 52-55	A1		C4	D2		C0			A1		C4	D2		C0			V 4-7	V 20-23	V 36-39	V 52-55
SP 2	XY 8-11	XY 24-27	XY 40-43	XY 56-59	A2		C5			C1		E0	A2		C5			C1		E0	V 8-11	V 24-27	V 40-43	V 56-59
SP 3	XY 12-15	XY 28-31	XY 44-47	XY 60-63					B0	C2	D0	E1					B0	C2	D0	E1	V 12-15	V 28-31	V 44-47	V 60-63
		XY						P1								P2						VTX		

Figure 6: Interpolation timing diagram

Above is an example of a tile the sequencer might receive from the SC. The write side is how the data get stacked into the XY and IJ buffers, the read side is how the data is passed to the GPRs. The IJ information is packed in the IJ buffer 4 quads at a time or two clocks. The sequencer allows at any given time as many as four quads to interpolate a parameter. They all have to come from the same primitive. Then the sequencer controls the write mask to the GPRs to write the valid data in.

3. Instruction Store

There is going to be only one instruction store for the whole chip. It will contain 4096 instructions of 96 bits each.

It is likely to be a 1 port memory; we use 1 clock to load the ALU instruction, 1 clocks to load the Fetch instruction, 1 clock to load 2 control flow instructions and 1 clock to write instructions.

The instruction store is loaded by the CP thru the register mapped registers.

The VS_BASE and PS_BASE context registers are used to specify for each context where its shader is in the instruction memory.

For the Real time commands the story is quite the same but for some small differences. There are no wrap-around points for real time so the driver must be careful not to overwrite regular shader data. The shared code (shared subroutines) uses the same path as real time.

4. Sequencer Instructions

All control flow instructions and move instructions are handled by the sequencer only. The ALUs will perform NOPs during this time (MOV PV,PV, PS,PS) if they have nothing else to do.

5. Constant Stores

5.1 Memory organizations

A likely size for the ALU constant store is 1024x128 bits. The read BW from the ALU constant store is 128 bits/clock and the write bandwidth is 32 bits/clock (directed by the CP bus size not by memory ports).

The maximum logical size of the constant store for a given shader is 256 constants. Or 512 for the pixel/vertex shader pair. The size of the re-mapping table is 128 lines (each line addresses 4 constants). The write granularity is 4 constants or 512 bits. It takes 16 clocks to write the four constants. Real time requires 256 lines in the physical memory (this is physically register mapped).

The texture state is also kept in a similar memory. The size of this memory is 320x96 bits (128 texture states for regular mode, 32 states for RT). The memory thus holds 128 texture states (192 bits per state). The logical size exposes 32 different states total, which are going to be shared between the pixel and the vertex shader. The size of the re-mapping table to for the texture state memory is 32 lines (each line addresses 1 texture state lines in the real memory). The CP write granularity is 1 texture state lines (or 192 bits). The driver sends 512 bits but the CP ignores the top 320 bits. It thus takes 6 clocks to write the texture state. Real time requires 32 lines in the physical memory (this is physically register mapped).

The control flow constant memory doesn't sit behind a renaming table. It is register mapped and thus the driver must reload its content each time there is a change in the control flow constants. Its size is 320*32 because it must hold 8 copies of the 32 dwords of control flow constants and the loop construct constants must be aligned.

The constant re-mapping tables for texture state and ALU constants are logically register mapped for regular mode and physically register mapped for RT operation.

5.2 Management of the Control Flow Constants

The control flow constants are register mapped, thus the CP writes to the according register to set the constant, the SQ decodes the address and writes to the block pointed by its current base pointer (CF_WR_BASE). On the read side, one level of indirection is used. A register (SQ_CONTEXT_MISC.CF_RD_BASE) keeps the current base pointer to the control flow block. This register is copied whenever there is a state change. Should the CP write to CF after the state change, the base register is updated with the (current pointer number +1)% number of states. This way, if the CP doesn't write to CF the state is going to use the previous CF constants.

5.3 Management of the re-mapping tables

5.3.1 *R400 Constant management*

The sequencer is responsible to manage two re-mapping tables (one for the constant store and one for the texture state). On a state change (by the driver), the sequencer will broadside copy the contents of its re-mapping tables to a new one. We have 8 different re-mapping tables we can use concurrently.

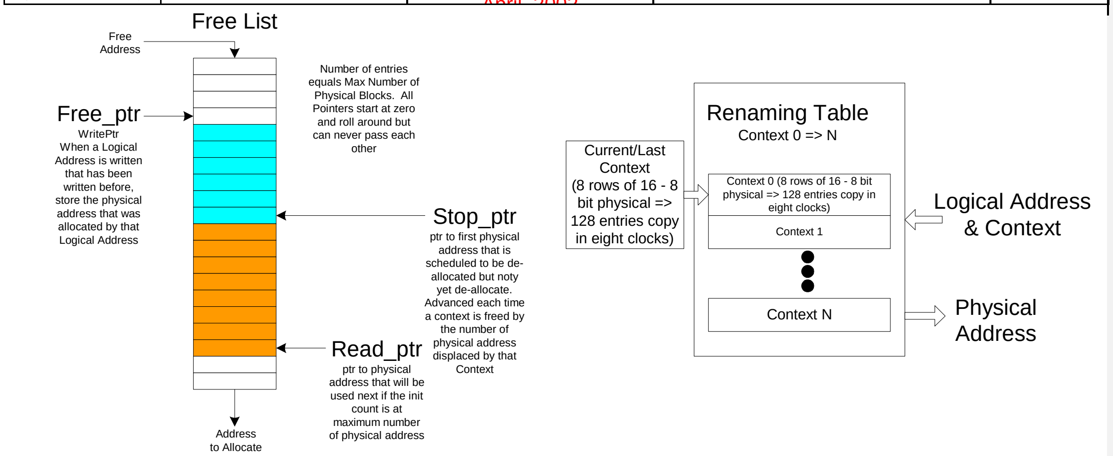
The constant memory update will be incremental, the driver only need to update the constants that actually changed between the two state changes.

For this model to work in its simplest form, the requirement is that the physical memory MUST be at least twice as large as the logical address space + the space allocated for Real Time. In our case, since the logical address space is 512 and the reserved RT space can be up to 256 entries, the memory must be of sizes 1280 and above. Similarly the size of the texture store must be of 32*2+32 = 96 entries and above.

5.3.2 *Proposal for R400LE constant management*

To make this scheme work with only 512+256 = 768 entries, upon reception of a CONTROL packet of state + 1, the sequencer would check for SQ_IDLE and PA_IDLE and if both are idle will erase the content of state to replace it with the new state (this is depicted in Figure 8: De-allocation mechanism~~Figure 9: De-allocation mechanism~~). Note that in the case a state is cleared a value of 0 is written to the corresponding de-allocation counter location so that when the SQ is going to report a state change, nothing will be de-allocated upon the first report.

The second path sets all context dirty bits that were used in the current state to 1 (thus allowing the new state to reuse these physical addresses if needed).



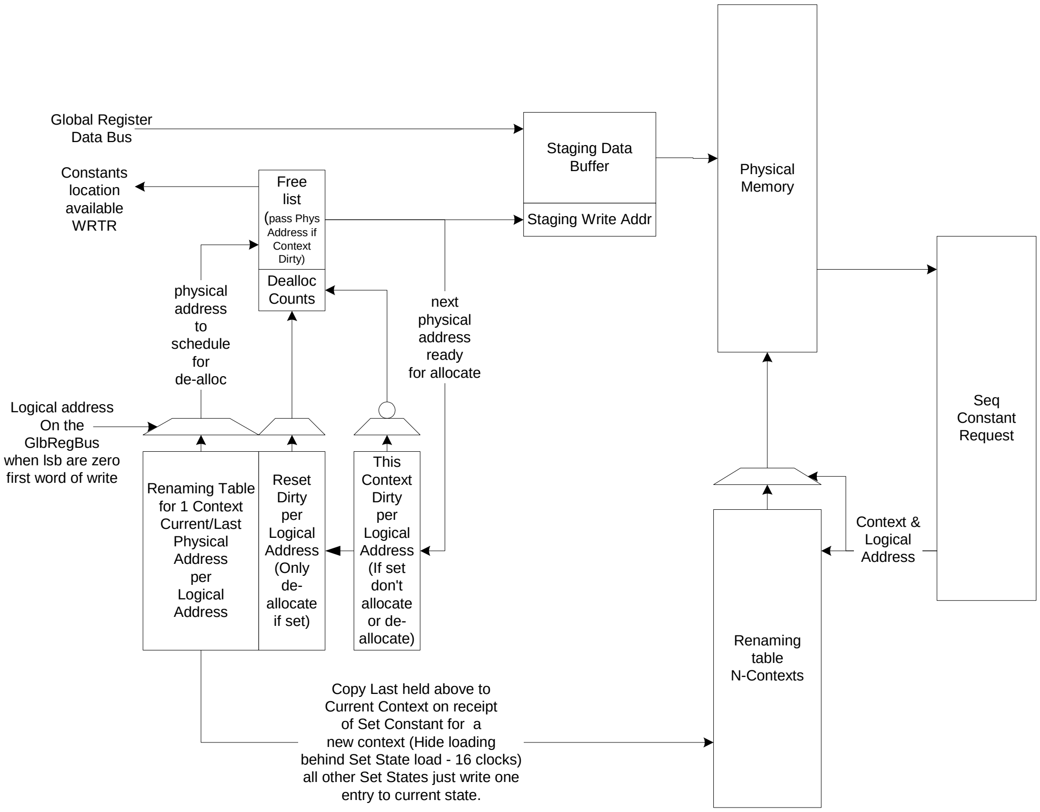


Figure 7~~8~~: Constant management

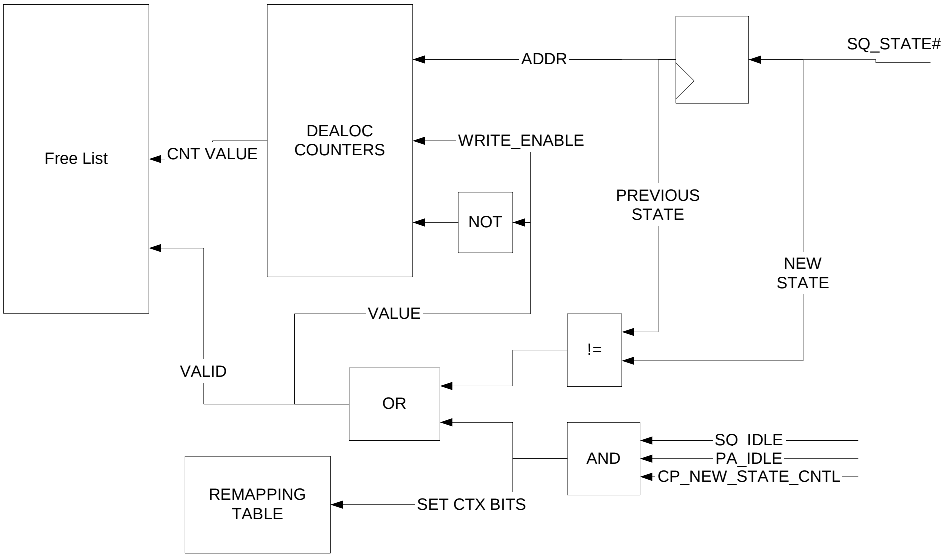


Figure 8~~9~~: De-allocation mechanism for R400LE

5.3.3 *Dirty bits*

Two sets of dirty bits will be maintained per logical address. The first one will be set to zero on reset and set when the logical address is addressed. The second one will be set to zero whenever a new context is written and set for each address written while in this context. The reset dirty is not set, then writing to that logical address will not require de-allocation of whatever address stored in the renaming table. If it is set and the context dirty is not set, then the physical address store needs to be de-allocated and a new physical address is necessary to store the incoming data. If they are both set, then the data will be written into the physical address held in the renaming for the current logical address. No de-allocation or allocation takes place. This will happen when the driver does a set constant twice to the same logical address between context changes. NOTE: It is important to detect and prevent this, failure to do it will allow multiple writes to allocate all physical memory and thus hang because a context will not fit for rendering to start and thus free up space.

5.3.4 *Free List Block*

A free list block that would consist of a counter (called the IFC or Initial Free Counter) that would reset to zero and incremented every time a chunk of physical memory is used until they have all been used once. This counter would be checked each time a physical block is needed, and if the original ones have not been used up, us a new one, else check the free list for an available physical block address. The count is the physical address for when getting a chunk from the counter.

Storage of a free list big enough to store all physical block addresses.

Maintain three pointers for the free list that are reset to zero. The first one we will call write_ptr. This pointer will identify the next location to write the physical address of a block to be de-allocated. Note: we can never free more physical memory locations than we have. Once recording address the pointer will be incremented to walk the free list like a ring.

The second pointer will be called stop_ptr. The stop_ptr pointer will be advanced by the number of address chunks de-allocates when a context finishes. The address between the stop_ptr and write_ptr cannot be reused because they are still in use. But as soon as the context using then is dismissed the stop_ptr will be advanced.

The third pointer will be called read_ptr. This pointer will point will point to the next address that can be used for allocation as long as the read_ptr does not equal the stop_ptr and the IFC is at its maximum count.

5.3.5 *De-allocate Block*

This block will maintain a free physical address block count for each context. While in current context, a count shall be maintained specifying how many blocks were written into the free list at the write_ptr pointer. This count will be reset upon reset or when this context is active on the back and different than the previous context. It is actually a count of blocks in the previous context that will no longer be used. This count will be used to advance the write_ptr pointer to make available the set of physical blocks freed when the previous context was done. This allows the discard or de-allocation of any number of blocks in one clock.

5.3.6 *Operation of Incremental model*

The basic operation of the model would start with the write_ptr, stop_ptr, read_ptr pointers in the free list set to zero and the free list counter is set to zero. Also all the dirty bits and the previous context will be initialized to zero. When the first set constants happen, the reset dirty bit will not be set, so we will allocate a physical location from the free list counter because its not at the max value. The data will be written into physical address zero. Both the additional copy of the renaming table and the context zeros of the big renaming table will be updated for the logical address that was written by set start with physical address of 0. This process will be repeated for any logical address that are not dirty until the context changes. If a logical address is hit that has its dirty bits set while in the same context, both dirty bits would be set, so the new data will be over-written to the last physical address assigned for this logical address. When the first draw command of the context is detected, the previous context stored in the additional renaming table will be copied to the larger renaming table in the current (new) context location. Then the set constant logical address with be loaded with a new physical address during the copy and if the reset dirty was set, the physical address it replaced in the renaming table would be entered at the write_ptr pointer location on the free list and the write_ptr will be incremented. The de-allocation counter for the previous context (eight) will be incremented. This as set states come in for this context one of the following will happen:

1.) No dirty bits are set for the logical address being updated. A line will be allocated of the free-list counter or the free list at read_ptr pointer if read_ptr != to stop_ptr .
2.) Reset dirty set and Context dirty not set. A new physical address is allocated, the physical address in the renaming table is put on the free list at write_ptr and it is incremented along with the de-allocate counter for the last context.
3.) Context dirty is set then the data will be written into the physical address specified by the logical address.

This process will continue as long as set states arrive. This block will provide backpressure to the CP whenever he has not free list entries available (counter at max and stop_ptr == read_ptr). The command stream will keep a count of contexts of constants in use and prevent more than max constants contexts from being sent.

Whenever a draw packet arrives, the content of the re-mapping table is written to the correct re-mapping table for the context number. Also if the next context uses less constants than the current one all exceeding lines are moved to the free list to be de-allocated later. This happens in parallel with the writing of the re-mapping table to the correct memory.

Now preferable when the constant context leaves the last ALU clause it will be sent to this block and compared with the previous context that left. (Init to zero) If they differ than the older context will no longer be referenced and thus can be de-allocated in the physical memory. This is accomplished by adding the number of blocks freed this context to the stop_ptr pointer. This will make all the physical addresses used by this context available to the read_ptr allocate pointer for future allocation.

This device allows representation of multiple contexts of constants data with N copies of the logical address space. It also allows the second context to be represented as the first set plus some new additional data by just storing the delta's. It allows memory to be efficiently used and when the constants updates are small it can store multiple context. However, if the updates are large, less contexts will be stored and potentially performance will be degraded. Although it will still perform as well as a ring could in this case.

5.4 Constant Store Indexing

In order to do constant store indexing, the sequencer must be loaded first with the indexes (that come from the GPRs). There are 144 wires from the exit of the SP to the sequencer (9 bits pointers x 16 vertexes/clock). Since the data must pass thru the Shader pipe for the float to fixed conversion, there is a latency of 4 clocks (1 instruction)

between the time the sequencer is loaded and the time one can index into the constant store. The assembly will look like this

```
MOVA  R1.X,R2.X       // Loads the sequencer with the content of R2.X, also copies the content of R2.X into R1.X
NOP                   // latency of the float to fixed conversion
ADD   R3,R4,C0[R2.X]// Uses the state from the sequencer to add R4 to C0[R2.X] into R3
```

Note that we don't really care about what is in the brackets because we use the state from the MOVA instruction. R2.X is just written again for the sake of simplicity and coherency.

The storage needed in the sequencer in order to support this feature is 2*64*9 bits = 1152 bits.

5.5 Real Time Commands

The real time commands constants are written by the CP using the register mapped registers allocated for RT. It works is the same way than when dealing with regular constant loads BUT in this case the CP is not sending a logical address but rather a physical address and the reads are not passing thru the re-mapping table but are directly read from the memory. The boundary between the two zones is defined by the CONST_EO_RT control register. Similarly, for the fetch state, the boundary between the two zones is defined by the TSTATE_EO_RT control register.

5.6 Constant Waterfalling

In order to have a reasonable performance in the case of constant store indexing using the address register, we are going to have the possibility of using the physical memory port for read only. This way we can read 1 constant per clock and thus have a worst-case waterfall mode of 1 vertex per clock. There is a small synchronization issue related with this as we need for the SQ to make sure that the constants where actually written to memory (not only sent to the sequencer) before it can allow the first vector of pixels or vertices of the state to go thru the ALUs. To do so, the sequencer keeps 8 bits (one per render state) and sets the bits whenever the last render state is written to memory and clears the bit whenever a state is freed.

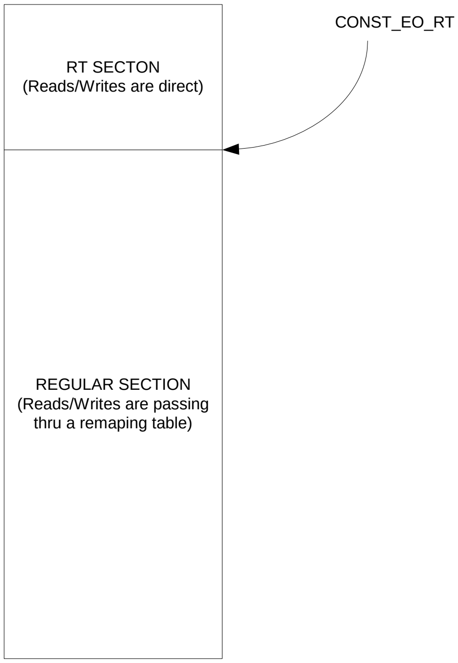

CONST_EO_RT
RT SECTON
(Reads/Writes are direct)
REGULAR SECTION
(Reads/Writes are passing thru a remaping table)

Figure 9~~10~~: The instruction store

6. Looping and Branches

Loops and branches are planned to be supported and will have to be dealt with at the sequencer level. We plan on supporting constant loops and branches using a control program.

6.1 The controlling state.

The R400 controling state consists of:

Boolean[256:0]
Loop_count[7:0][31:0]
Loop_Start[7:0][31:0]
Loop_Step[7:0][31:0]

That is 256 Booleans and 32 loops.

We have a stack of 4 elements for nested calls of subroutines and 4 loop counters to allow for nested loops.

This state is available on a per shader program basis.

6.2 The Control Flow Program

We'd like to be able to code up a program of the form:

```
1:      Loop
2:      Exec    TexFetch
3:              TexFetch
4:              ALU
5:              ALU
6:              TexFetch
7:      End Loop
8:      ALU Export
```

But realize that 3: may be dependent on 2: and 4: is almost certainly dependent on 2: and 3:. Without clausing, these dependencies need to be expressed in the Control Flow instructions. Additionally, without separate 'texture clauses' and 'ALU clauses' we need to know which instructions to dispatch to the Texture Unit and which to the ALU unit. This information will be encapsulated in the flow control instructions.

Each control flow instruction will contain 2 bits of information for each (non-control flow) instruction:

a) ALU or Texture
b) Serialize Execution

(b) would force the thread to stop execution at this point (before the instruction is executed) and wait until all textures have been fetched. Given the allocation of reserved bits, this would mean that the count of an 'Exec' instruction would be limited to about 8 (non-control-flow) instructions. If more than this were needed, a second Exec (with the same conditions) would be issued.

Another function that relies upon 'clauses' is allocation and order of execution. We need to assure that pixels and vertices are exported in the correct order (even if not all execution is ordered) and that space in the output buffers are allocated in order. Additionally data can't be exported until space is allocated. A new control flow instruction:

Alloc <buffer select -- position,parameter, pixel or vertex memory. And the size required>.

would be created to mark where such allocation needs to be done. To assure allocation is done in order, the actual allocation for a given thread can not be performed unless the equivalent allocation for all previous threads is already completed. The implementation would also assure that execution of instruction(s) following the serialization due to the Alloc will occur in order -- at least until the next serialization or change from ALU to Texture. In most cases this will allow the exports to occur without any further synchronization. Only 'final' allocations or position allocations are

guaranteed to be ordered. Because strict ordering is required for pixels, parameters and positions, this implies only a single alloc for these structures. Vertex exports to memory do not require ordering during allocation and so multiple 'allocs' may be done.

6.2.1 *Control flow instructions table*

Formatted: Bullets and Numbering

Here is the revised control flow instruction set.

Note that whenever a field is marked as RESERVED, it is assumed that all the bits of the field are cleared (0).

Execute					
47	46… 43	40 … 34	33 …16	15…12	11 … 0
Addressing	0001	RESERVED	Instructions type + serialize (9 instructions)	Count	Exec Address

Execute up to 9 instructions at the specified address in the instruction memory. The Instruction type field tells the sequencer the type of the instruction (LSB) (1 = Texture, 0 = ALU and whether to serialize or not the execution (MSB) (1 = Serialize, 0 = Non-Serialized).

NOP		
47	46 … 43	42 … 0
Addressing	0010	RESERVED

This is a regular NOP.

Conditional_Execute						
47	46 … 43	42	41 … 34	33…16	15 …12	11 … 0
Addressing	0011	Condition	Boolean address	Instructions type + serialize (9 instructions)	Count	Exec Address

If the specified Boolean (8 bits can address 256 Booleans) meets the specified condition then execute the specified instructions (up to 9 instructions). If the condition is not met, we go on to the next control flow instruction.

Conditional_Execute_Predicates							
47	46 … 43	42	41 … 36	35 … 34	33…16	15…12	11 … 0
Addressing	0010	Condition	RESERVED	Predicate vector	Instructions type + serialize (9 instructions)	Count	Exec Address

Check the AND/OR of all current predicate bits. If AND/OR matches the condition execute the specified number of instructions. We need to AND/OR this with the kill mask in order not to consider the pixels that aren't valid. If the condition is not met, we go on to the next control flow instruction.

Loop_Start				
47	46 … 43	42 … 17	16 … 12	11 … 0
Addressing	0101	RESERVED	loop ID	Jump address

Loop Start. Compares the loop iterator with the end value. If loop condition not met jump to the address. Forward jump only. Also computes the index value. The loop id must match between the start to end, and also indicates which control flow constants should be used with the loop.

Loop_End					
47	46 … 43	42 … 20~~17~~	19… 17	16 … 12	11 … 0
Addressing	0011	RESERVED	Predicate break	loop ID	start address

Loop end. Increments the counter by one, compares the loop count with the end value. If loop condition met, continue, else, jump BACK to the start of the loop. If predicate break != 0, then compares predicate vector n (specified by predicate break number). If all bits cleared then break the loop.

The way this is described does not prevent nested loops, and the inclusion of the loop id make this easy to do.

Conditionnal_Call						
47	46 … 43	42	41 …37	35 … 34	33 … 12	11 … 0
Addressing	0111	Condition	RESERVED	Predicate vector	RESERVED	Jump address

If the condition is met, jumps to the specified address and pushes the control flow program counter on the stack.

Return		
47	46 … 43	42 … 0
Addressing	1000	RESERVED

Pops the topmost address from the stack and jumps to that address. If nothing is on the stack, the program will just continue to the next instruction.

Conditionnal_Jump						
47	46 … 43	42	41… 34	33	32 … 12	11 … 0
Addressing	1001	Condition	Boolean address	FW only	RESERVED	Jump address

Allocate				
47	46 … 43	42…41	40 … 4	3 …0
Debug	1010	Buffer Select	RESERVED	Allocation size

Buffer Select takes a value of the following:
01 – position export (ordered export)
10 – parameter cache or pixel export (ordered export)
11 – pass thru (out of order exports).

If debug is set this is a debug alloc (ignore if debug DB_ON register is set to off).

End Of Program		
47	46 … 43	42… 0
RESERVED	1011	RESERVED

Marks the end of the program.

6.3 Implementation

Formatted: Bullets and Numbering

The envisioned implementation has a buffer that maintains the state of each thread. A thread lives in a given location in the buffer during its entire life, but the buffer has FIFO qualities in that threads leave in the order that they enter. Actually two buffers are maintained -- one for Vertices and one for Pixels. The intended implementation would allow for:

16 entries for vertices
48 entries for pixels.

From each buffer, arbitration logic attempts to select 1 thread for the texture unit and 1 (interleaved) thread for the ALU unit. Once a thread is selected it is read out of the buffer, marked as invalid, and submitted to appropriate execution unit. It is returned to the buffer (at the same place) with its status updated once all possible sequential instructions have been executed. A switch from ALU to TEX or visa-versa or a Serialize_Execution modifier forces the thread to be returned to the buffer.

Each entry in the buffer will be stored across two physical pieces of memory - most bits will be stored in a 1 read port device. Only bits needed for thread arbitration will be stored in a highly multi-ported structure. The bits kept in the 1 read port device will be termed 'state'. The bits kept in the multi-read ported device will be termed 'status'.

'State Bits' needed include:

1. Control Flow Instruction Pointer (12 bits),
2. Execution Count Marker 4 bits),
3. Loop Iterators (4x9 bits),
4. Call return pointers (4x12 bits),
5. Predicate Bits(4x64 bits),
6. Export ID (1 bit),
7. Parameter Cache base Ptr (7 bits),
8. GPR Base Ptr (8 bits),
9. Context Ptr (3 bits).
10. LOD corrections (6x16 bits)

Formatted: Bullets and Numbering

Absent from this list are 'Index' pointers. These are costly enough that I'm presuming that they are instead stored in the GPRs. The first seven fields above (Control Flow Ptr, Execution Count, Loop Counts, call return ptrs, Predicate bits, PC base ptr and export ID) are updated every time the thread is returned to the buffer based on how much progress has been mode on thread execution. GPR Base Ptr, Context Ptr and LOD corrections are unchanged throughout execution of the thread.

'Status Bits' needed include:

- Valid Thread
- Texture/ALU engine needed
- Texture Reads are outstanding
- Waiting on Texture Read to Complete
- Allocation Wait (2 bits)
- 00 – No allocation needed
- 01 – Position export allocation needed (ordered export)
- 10 – Parameter or pixel export needed (ordered export)
- 11 – pass thru (out of order export)
- Allocation Size (4 bits)
- Position Allocated
- First thread of a new context
- Event thread (NULL thread that needs to trickle down the pipe)
- Last (1 bit)

Formatted: Bullets and Numbering

All of the above fields from all of the entries go into the arbitration circuitry. The arbitration circuitry will select a winner for both the Texture Engine and for the ALU engine. There are actually two sets of arbitration -- one for pixels and one for vertices. A final selection is then done between the two. But the rest of this implementation summary only considers the 'first' level selection which is similar for both pixels and vertices.

Texture arbitration requires no allocation or ordering so it is purely based on selecting the 'oldest' thread that requires the Texture Engine.

ALU arbitration is a little more complicated. First, only threads where either of Texture_Reads_outstanding or Waiting_on_Texture_Read_to_Complete are '0' are considered. Then if Allocation_Wait is active, these threads are further filtered based on whether space is available. If the allocation is position allocation, then the thread is only considered if all 'older' threads have already done their position allocation (position allocated bits set). If the allocation is parameter or pixel allocation, then the thread is only considered if it is the oldest thread. Also a thread is not considered if it is a parameter or pixel or position allocation, has its First_thread_of_a_new_context bit set and would cause ALU interleaving with another thread performing the same parameter or pixel or position allocation. Finally the 'oldest' of the threads that pass through the above filters is selected. If the thread needed to allocate, then at this time the allocation is done, based on Allocation_Size. If a thread has its “last” bit set, then it is also removed from the buffer, never to return.

If I now redefine 'clauses' to mean 'how many times the thread is removed from the thread buffer for the purpose of exection by either the ALU or Texture engine', then the minimum number of clauses needed is 2 -- one to perform the allocation for exports (execution automatically halts after an 'Alloc' instruction) (but doesn't performs the actual allocation) and one for the actual ALU/export instructions. As the 'Alloc' instruction could be part of a texture clause (presumably the final instruction in such a clause), a thread could still execute in this minimal number of 2 clauses, even if it involved texture fetching.

The Texture_Reads_Outstanding bit must be updated by the sequencer, based on keeping track of how many Texture Clauses have been executed by a given thread that have not yet had there data returned. Any number above 0 results in this bit being set. We could consider forcing synchronization such that two texture clauses for a given thread may not be outstanding at any time (that would be my preference for simplicity reasons and because it would require only very little change in the texture pipe interface). This would allow the sequencer to set the bit on execution of the texture clause, and allow the texture unit to return a pointer to the thread buffer on completion that clears the bit.

~~Examples of control flow programs are located in the R400 programming guide document.~~

~~The basic model is as follows:~~

~~The render state defined the clause boundaries:~~
~~Vertex_shader_fetch[7:0][7:0] // eight 8 bit pointers to the location where each clauses control program is located~~
~~Vertex_shader_alu[7:0][7:0] // eight 8 bit pointers to the location where each clauses control program is located~~
~~Pixel_shader_fetch[7:0][7:0] // eight 8 bit pointers to the location where each clauses control program is located~~
~~Pixel_shader_alu[7:0][7:0] // eight 8 bit pointers to the location where each clauses control program is located~~

~~**A pointer value of FF means that the clause doesn’t contain any instructions**.~~

~~The control program for a given clause is executed to completion before moving to another clause, (with the exception of the pick two nature of the alu execution). The control program is the only program aware of the clause boundaries.~~

~~The control program has nine basic instructions:~~

~~Execute~~
~~Conditional_execute~~
~~Conditional_Execute_Predicates~~
~~Conditional_jump~~
~~Conditionnal_Call~~
~~Return~~
~~Loop_start~~
~~Loop_end~~
~~NOP~~

~~Execute, causes the specified number of instructions in instruction store to be executed.~~
~~Conditional_execute checks a condition first, and if true, causes the specified number of instructions in instruction store to be executed.~~
~~Loop_start resets the corresponding loop counter to the start value on the first pass after it checks for the end condition and if met jumps over to a specified address.~~
~~Loop_end increments (decrements?) the loop counter and jumps back the specified number of instructions.~~
~~Conditionnal_Call jumps to an address and pushes the IP counter on the stack if the condition is met. On the return instruction, the IP is popped from the stack.~~

~~Conditional_execute_Predicates executes a block of instructions if all bits in the predicate vectors meet the condition.~~
~~Conditional_jumps jumps to an address if the condition is met.~~
~~NOP is a regular NOP~~

~~NOTE THAT ALL JUMPS MUST JUMP TO EVEN CFP ADDRESSES since there are two control flow instructions per memory line. Thus the compiler must insert NOPs where needed to align the jumps on even CFP addresses.~~

~~Also if the jump is logically bigger than pshader_cntl_size (or vshader_cntl_size) we break the program (clause) and set the debug registers. If an execute or conditional_execute is lower than cntl_size or bigger than size we also break the program (clause) and set the debug registers.~~

~~We have to fit instructions into 48 bits in order to be able to put two control flow instruction per line in the instruction store.~~

~~A value of 1 in the Addressing means that the address specified in the Exec Address field (or in the jump address field) is an ABSOLUTE address. If the addressing field is cleared (should be the default) then the address is relative to the base of the current shader program.~~

~~**Note that whenever a field is marked as RESERVED, it is assumed that all the bits of the field are cleared (0).**~~

~~Execute up to 4k instructions at the specified address in the instruction memory. If Last is set, this is the last group of instructions of the clause.~~

~~This is a regular NOP. If Last is set, this is the last instruction of the clause.~~

~~If the specified Boolean (8 bits can address 256 Booleans) meets the specified condition then execute the specified instructions (up to 4k instructions). If Last is set, then if the condition is met, this is the last group of instructions to be executed in the clause. If the condition is not met, we go on to the next control flow instruction.~~

~~Check the AND/OR of all current predicate bits. If AND/OR matches the condition execute the specified number of instructions. We need to AND/OR this with the kill mask in order not to consider the pixels that aren't valid. If Last is set, then if the condition is met, this is the last group of instructions to be executed in the clause. If the condition is not met, we go on to the next control flow instruction.~~

~~Loop Start. Compares the loop iterator with the end value. If loop condition not met jump to the address. Forward jump only. Also computes the index value. The loop id must match between the start to end, and also indicates which control flow constants should be used with the loop.~~

~~Loop end. Increments the counter by one, compares the loop count with the end value. If loop condition met, continue, else, jump BACK to the start of the loop.~~

~~The way this is described does not prevent nested loops, and the inclusion of the loop id make this easy to do.~~

~~If the condition is met, jumps to the specified address and pushes the control flow program counter on the stack.~~

~~Pops the topmost address from the stack and jumps to that address. If nothing is on the stack, the program will just continue to the next instruction.~~

~~If condition met, jumps to the address. FORWARD jump only allowed if bit 31 set. Bit 31 is only an optimization for the compiler and should NOT be exposed to the API.~~

~~To prevent infinite loops, we will keep 9 bits loop iterators instead of 8 (we are only able to loop 256 times). If the counter goes higher than 255 then the loop_end or the loop_start instruction is going to break the loop and set the debug GPRs.~~

Formatted: Bullets and Numbering

~~6.3~~6.4 Data dependant predicate instructions

Data dependant conditionals will be supported in the R400. The only way we plan to support those is by supporting three vector/scalar predicate operations of the form:

PRED_SETE_# - similar to SETE except that the result is 'exported' to the sequencer.
PRED_SETNE_# - similar to SETNE except that the result is 'exported' to the sequencer.
PRED_SETGT_# - similar to SETGT except that the result is 'exported' to the sequencer
PRED_SETGTE_# - similar to SETGTE except that the result is 'exported' to the sequencer

For the scalar operations only we will also support the two following instructions:
PRED_SETE0_# – SETE0
PRED_SETE1_# – SETE1

The export is a single bit - 1 or 0 that is sent using the same data path as the MOVA instruction. The sequencer will maintain 4 sets of 64 bit predicate vectors (in fact 8 sets because we interleave two programs but only 4 will be exposed) and use it to control the write masking. This predicate is not maintained across clause boundaries. The # sign is used to specify which predicate set you want to use 0 thru 3.

Then we have two conditional execute bits. The first bit is a conditional execute “on” bit and the second bit tells us if we execute on 1 or 0. For example, the instruction:

P0_ADD_# R0,R1,R2

Is only going to write the result of the ADD into those GPRs whose predicate bit is 0. Alternatively, P1_ADD_# would only write the results to the GPRs whose predicate bit is set. The use of the P0 or P1 without precharging the sequencer with a PRED instruction is undefined.

{Issue: do we have to have a NOP between PRED and the first instruction that uses a predicate?}

Formatted: Bullets and Numbering

~~6.4~~6.5 HW Detection of PV,PS

Because of the control program, the compiler cannot detect statically dependant instructions. In the case of non-masked writes and subsequent reads the sequencer will insert uses of PV,PS as needed. This will be done by comparing the read address and the write address of consecutive instructions. For masked writes, the sequencer will insert NOPs wherever there is a dependant read/write.

The sequencer will also have to insert NOPs between PRED_SET and MOVA instructions and their uses.

Formatted: Bullets and Numbering

~~6.5~~6.6 Register file indexing

Because we can have loops in fetch clause, we need to be able to index into the register file in order to retrieve the data created in a fetch clause loop and use it into an ALU clause. The instruction will include the base address for register indexing and the instruction will contain these controls:

Bit7	Bit 6	
0	0	'absolute register'
0	1	'relative register'
1	0	'previous vector'
1	1	'previous scalar'

In the case of an absolute register we just take the address as is. In the case of a relative register read we take the base address and we add to it the loop_index and this becomes our new address that we give to the shader pipe.

The sequencer is going to keep a loop index computed as such:

Index = Loop_iterator*Loop_step + Loop_start.

We loop until loop_iterator = loop_count. Loop_step is a signed value [-128…127]. The computed index value is a 10 bit counter that is also signed. Its real range is [-256,256]. The tenth bit is only there so that we can provide an out of range value to the “indexing logic” so that it knows when the provided index is out of range and thus can make the necessary arrangements.
~~Predicated Instruction support for Texture clauses~~
~~For texture clauses, we support the following optimization: we keep 1 bit (thus 4 bits for the four predicate vectors) per predicate vector in the reservation stations. A value of 1 means that one ore more elements in the vector have a value of one (thus we have to do the texture fetches for the whole vector). A value of 0 means that no elements in the vector have his predicate bit set and we can thus skip over the texture fetch.~~ **~~We have to make sure the invalid pixels aren’t considered with this optimization.~~**

~~6.6~~6.7 Debugging the Shaders

In order to be able to debug the pixel/vertex shaders efficiently, we provide 2 methods.

~~6.6.1~~6.7.1 *Method 1: Debugging registers*

Current plans are to expose 2 debugging, or error notification, registers:
1. address register where the first error occurred
2. count of the number of errors

The sequencer will detect the following groups of errors:
- count overflow
- constant indexing overflow
- register indexing overflow

Compiler recognizable errors:
- jump errors
  relative jump address > size of the control flow program
- call stack
  call with stack full
  return with stack empty

A jump error will always cause the program to break. In this case, a break means that a clause will halt execution, but allowing further clauses to be executed.

With all the other errors, program can continue to run, potentially to worst-case limits. The program will only break if the DB_PROB_BREAK register is set.

If indexing outside of the constant or the register range, causing an overflow error, the hardware is specified to return the value with an index of 0. This could be exploited to generate error tokens, by reserving and initializing the 0th register (or constant) for errors.

{ISSUE : Interrupt to the driver or not?}

~~6.6.2~~6.7.2 *Method 2: Exporting the values in the GPRs ~~(12)~~*

The sequencer will have a debug active, count register and an address register for this mode~~ and 3 bits per clause specifying the execution mode for each clause~~.~~ The modes can be :~~
~~Normal~~

~~2)Debug Kill~~
~~2)~~1)~~Debug Addr + Count~~

Formatted: Bullets and Numbering

Under the normal mode execution follows the normal course. ~~Under the kill mode, all control flow instructions are executed but all normal shader instructions of the clause are replaced by NOPs. Only debug_export instructions of clause 7 will be executed under the debug kill setting. Under the other mode, normal execution is done until we reach an address specified by the address register and instruction count (useful for loops) specified by the count register. After we have hit the instruction n times (n=count) we switch the clause to the kill mode.~~

Under the debug mode~~ (debug kill OR debug Addr + count),~~ it is assumed that the program~~clause 7~~ is always exporting ~~12~~ n debug vectors and that all other exports to the SX block (position, color, z, ect) will been turned off (changed into NOPs) by the sequencer (even if they occur before the address stated by the ADDR debug register).

7. Pixel Kill Mask

A vector of 64 bits is kept by the sequencer per group of pixels/vertices. Its purpose is to optimize the texture fetch requests and allow the shader pipe to kill pixels using the following instructions:

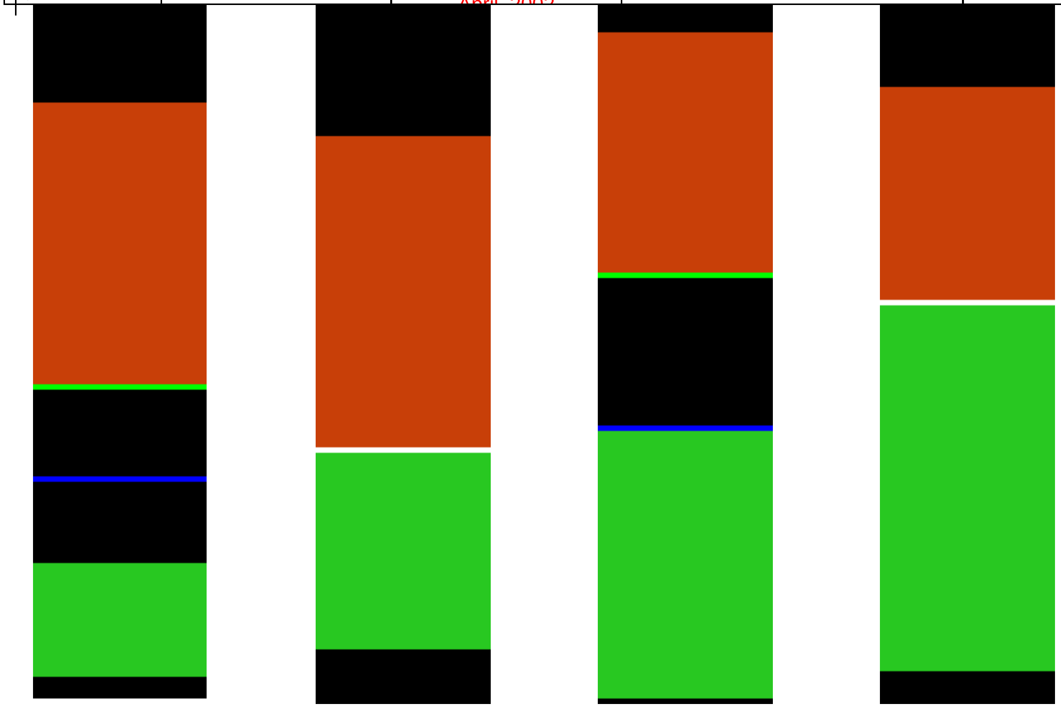
MASK_SETE
MASK_SETNE
MASK_SETGT
MASK_SETGTE

8. Multipass vertex shaders (HOS)

Multipass vertex shaders are able to export from the 6 last clauses but to memory ONLY.

9. Register file allocation

The register file allocation for vertices and pixels can either be static or dynamic. In both cases, the register file in managed using two round robins (one for pixels and one for vertices). In the dynamic case the boundary between pixels and vertices is allowed to move, in the static case it is fixed to 128-VERTEX_REG_SIZE for vertices and PIXEL_REG_SIZE for pixels.



Above is an example of how the algorithm works. Vertices come in from top to bottom; pixels come in from bottom to top. Vertices are in orange and pixels in green. The blue line is the tail of the vertices and the green line is the tail of the pixels. Thus anything between the two lines is shared. When pixels meets vertices the line turns white and the boundary is static until both vertices and pixels share the same “unallocated bubble”. Then the boundary is allowed to move again. The numbering of the GPRs starts from the bottom of the picture at index 0 and goes up to the top at index 127.

10. Fetch Arbitration

The fetch arbitration logic chooses one of the 8 potentially pending fetch clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute. Once chosen, the clause state machine will send one 2x2 fetch per clock (or 4 fetches in one clock every 4 clocks) until all the fetch instructions of the clause are sent. This means that there cannot be any dependencies between two fetches of the same clause.

The arbitrator will not wait for the fetches to return prior to selecting another clause for execution. The fetch pipe will be able to handle up to X(?) in flight fetches and thus there can be a fair number of active clauses waiting for their fetch return data.

11. ALU Arbitration

ALU arbitration proceeds in almost the same way than fetch arbitration. The ALU arbitration logic chooses one of the 8 potentially pending ALU clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute. There are two ALU arbiters, one for the even clocks and one for the odd clocks. For example, here is the sequencing of two interleaved ALU clauses (E and O stands for Even and Odd sets of 4 clocks):

Einst0 Oinst0 Einst1 Oinst1 Einst2 Oinst2 Einst0 Oinst3 Einst1 Oinst4 Einst2 Oinst0…
Proceeding this way hides the latency of 8 clocks of the ALUs. Also note that the interleaving also occurs across clause boundaries.

12. Handling Stalls

When the output file is full, the sequencer prevents the ALU arbitration logic from selecting the last clause (this way nothing can exit the shader pipe until there is place in the output file. If the packet is a vertex packet and the position buffer is full (POS_FULL) then the sequencer also prevents a thread from entering the exporting clause (3?). The sequencer will set the OUT_FILE_FULL signal n clocks before the output file is actually full and thus the ALU arbiter will be able read this signal and act accordingly by not preventing exporting clauses to proceed.

13. Content of the reservation station FIFOs

The reservation FIFOs contain the state of the vector of pixels and vertices. We have two sets of those: one for pixels, and one for vertices. They contain 3 bits of Render State 7 bits for the base address of the GPRs, some bits for LOD correction and coverage mask information in order to fetch fetch for only valid pixels, the quad address.

14. The Output File

The output file is where pixels are put before they go to the RBs. The write BW to this store is 256 bits/clock. Just before this output file are staging registers with write BW 512 bits/clock and read BW 256 bits/clock. The staging registers are 4x128 (and there are 16 of those on the whole chip).

15. IJ Format

The IJ information sent by the PA is of this format on a per quad basis:

We have a vector of IJ's (one IJ per pixel at the centroid of the fragment or at the center of the pixel depending on the mode bit). The interpolation is done at a different precision across the 2x2. The upper left pixel's parameters are always interpolated at full 20x24 mantissa precision. Then the result of the interpolation along with the difference in IJ in reduced precision is used to interpolate the parameter for the other three pixels of the 2x2. Here is how we do it:

Assuming P0 is the interpolated parameter at Pixel 0 having the barycentric coordinates I(0), J(0) and so on for P1,P2 and P3. Also assuming that A is the parameter value at V0 (interpolated with I), B is the parameter value at V1 (interpolated with J) and C is the parameter value at V2 (interpolated with (1-I-J).

$$\Delta 01I = I(1) - I(0)$$
$$\Delta 01J = J(1) - J(0)$$
$$\Delta 02I = I(2) - I(0)$$
$$\Delta 02J = J(2) - J(0)$$
$$\Delta 03I = I(3) - I(0)$$
$$\Delta 03J = J(3) - J(0)$$

P0	P1
P2	P3

$$P0 = C + I(0) * (A - C) + J(0) * (B - C)$$
$$P1 = P0 + \Delta 01I * (A - C) + \Delta 01J * (B - C)$$
$$P2 = P0 + \Delta 02I * (A - C) + \Delta 02J * (B - C)$$
$$P3 = P0 + \Delta 03I * (A - C) + \Delta 03J * (B - C)$$

P0 is computed at 20x24 mantissa precision and P1 to P3 are computed at 8X24 mantissa precision. So far no visual degradation of the image was seen using this scheme.

Multiplies (Full Precision): 2
Multiplies (Reduced precision): 6
Subtracts 19x24 (Parameters): 2

Adds: 8

FORMAT OF P0's IJ : Mantissa 20 Exp 4 for I + Sign
Mantissa 20 Exp 4 for J + Sign

FORMAT of Deltas (x3):Mantissa 8 Exp 4 for I + Sign
Mantissa 8 Exp 4 for J + Sign

Total number of bits : 20*2 + 8*6 + 4*8 + 4*2 = 128

All numbers are kept using the un-normalized floating point convention: if exponent is different than 0 the number is normalized if not, then the number is un-normalized. The maximum range for the IJs (Full precision) is +/- 63 and the range for the Deltas is +/- 127.

15.1 Interpolation of constant attributes

Because of the floating point imprecision, we need to take special provisions if all the interpolated terms are the same or if two of the barycentric coordinates are the same.

We start with the premise that if A = B and B = C and C = A, then P0,1,2,3 = A. Since one or more of the IJ terms may be zero, so we extend this to:

```
if (A=B and B=C and C=A)
   P0,1,2,3 = A;
else if ((I = 0) or (J = 0)) and
         ((J = 0) or (1-I-J = 0)) and
         ((1-J-I = 0) or (I = 0))) {
               if(I != 0) {
                   P0 = A;
               } else if(J != 0) {
                   P0 = B;
               } else {
                   P0 = C;
               }
            //rest of the quad interpolated normally
}
else
{
            normal interpolation
}
```

16. Staging Registers

In order for the reuse of the vertices to be 14, the sequencer will have to re-order the data sent IN ORDER by the VGT for it to be aligned with the parameter cache memory arrangement. Given the following group of vertices sent by the VGT:

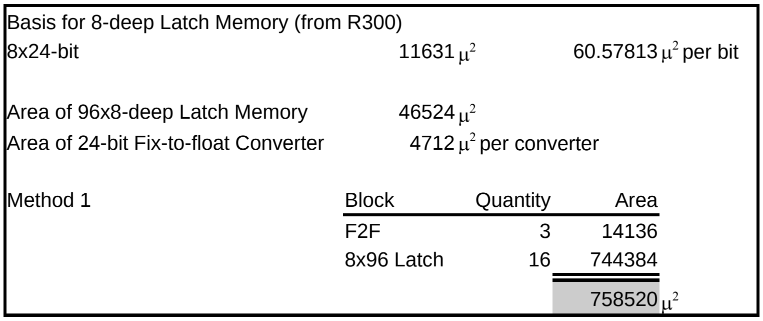
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 || 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 || 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 || 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63

The sequencer will re-arrange them in this fashion:

0 1 2 3 16 17 18 19 32 33 34 35 48 49 50 51 || 4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55 || 8 9 10 11 24 25 26 27 40 41 42 43 56 57 58 59 || 12 13 14 15 28 29 30 31 44 45 46 47 60 61 62 63

The || markers show the SP divisions. In the event a shader pipe is broken, the VGT will send padding to account for the missing pipe. For example, if SP1 is broken, vertices 4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55 will still be sent by the VGT to the SQ **BUT** will not be processed by the SP and thus should be considered invalid (by the SU and VGT).

The most straightforward, *non-compressed* interface method would be to convert, in the VGT, the data to 32-bit floating point prior to transmission to the VSISRs. In this scenario, the data would be transmitted to (and stored in) the VSISRs in full 32-bit floating point. This method requires three 24-bit fixed-to-float converters in the VGT. Unfortunately, it also requires and additional 3,072 bits of storage across the VSISRs. This interface is illustrated in Figure 11~~Figure 12~~. The area of the fixed-to-float converters and the VSISRs for this method is roughly estimated as 0.759sqmm using the R300 process. The gate count estimate is shown in Figure 10~~Figure 11~~.



Basis for 8-deep Latch Memory (from R300)			
8x24-bit	11631 μ^2		60.57813 μ^2 per bit
Area of 96x8-deep Latch Memory	46524 μ^2		
Area of 24-bit Fix-to-float Converter	4712 μ^2 per converter		
Method 1	Block	Quantity	Area
	F2F	3	14136
	8x96 Latch	16	744384
			758520 μ^2

Figure 10~~11~~:Area Estimate for VGT to Shader Interface

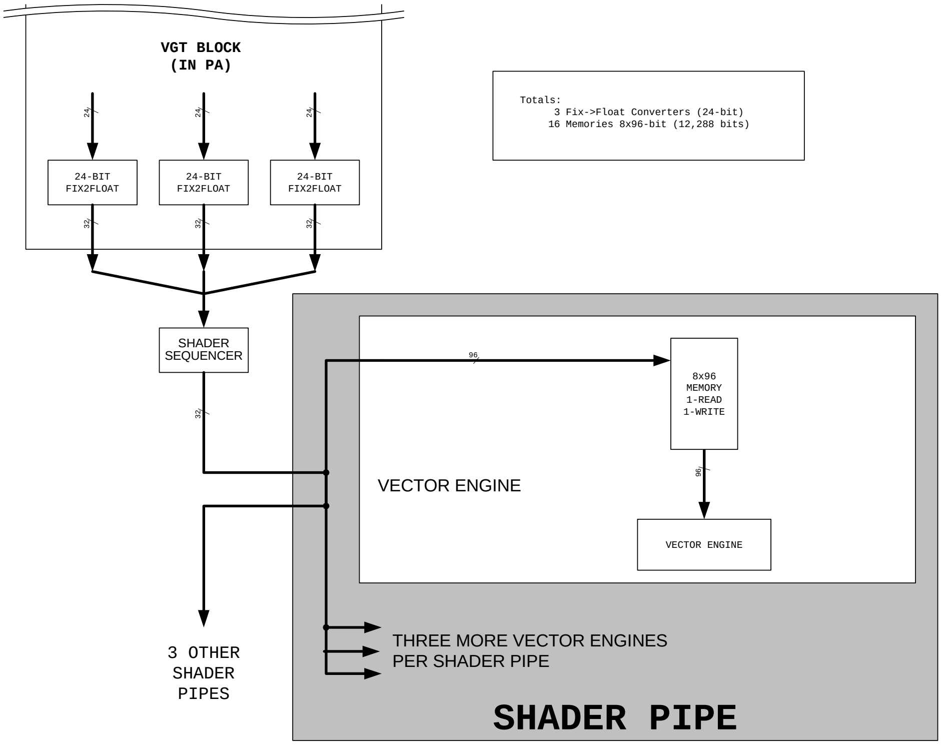


Figure 11~~12~~:VGT to Shader Interface

17. The parameter cache

The parameter cache is where the vertex shaders export their data. It consists of 16 128x128 memories (1R/1W). The reuse engine will make it so that all vertexes of a given primitive will hit different memories. The allocation method for these memories is a simple round robin. The parameter cache pointers are mapped in the following way: 4MSBs are the memory number and the 7 LSBs are the address within this memory.

MEMORY NUMBER 4 bits	ADDRESS 7 bits

The PA generates the parameter cache addresses as the positions come from the SQ. All it needs to do is keep a Current_Location pointer (7 bits only) and as the positions comes increment the memory number. When the memory number field wraps around, the PA increments the Current_Location by VS_EXPORT_COUNT~~_7~~ (a snooped register from the SQ). As an example, say the memories are all empty to begin with and the vertex shader is exporting 8 parameters per vertex (VS_EXPORT_COUNT~~_7~~ = 8). The first position received is going to have the PC address 00000000000 the second one 00010000000, third one 00100000000 and so on up to 11110000000. Then the next position received (the 17th) is going to have the address 00000001000, the 18th 00010001000, the 19th 00100001000 and so on. The Current_location is NEVER reset BUT on chip resets. The only thing to be careful about is that if the SX doesn't send you a full group of positions (<64) then you need to fill the address space so that the next group starts correctly aligned (for example if you receive only 33 positions then you need to add 2*VS_EXPORT_COUNT ~~_7~~to Current_Location and reset the memory count to 0 before the next vector begins).

17.1 Export restrictions

Formatted: Bullets and Numbering

17.1.1 *Pixel exports:*

Pixels can export 1,2,3 or 4 color buffers to the SX(+z). The exports will be done in order. The PRED_OPTIMIZE function has to be turned of if the exports are done using interleaved predicated instructions. The exports will always be ordered to the SX.

17.1.2 *Vertex exports:*

Formatted: Bullets and Numbering

Position or parameter caches can be exported in any order in the shader program. It is always better to export posistion as soon as possible. Position has to be exported in a single export block (no texture instructions can be placed between the exports). Parameter cache exports can be done in any order with texture instructions interleaved. The PRED_OPTIMIZE function has to be turned of if the exports are done using interleaved predicated instructions to the Parameter cache (see Arbitration restrictions for details). The exports will always be allocated in order to the SX.

17.1.3 *Pass thru exports:*

Formatted: Bullets and Numbering

Pass thru exports have to be done in groups of the form:

```
Alloc 4 (8 or 12)
Execute ALU(ADDR) ALU(DATA) ALU(DATA) ALU(DATA)…
```

They cannot have texture instructions interleaved in the export block. These exports **are not guaranteed to be ordered**.

Also, when doing a pass thru export, Position MUST be exported AFTER all pass thru exports. This position export is used to synchronize the chip when doing a transition from pass thru shader to regular shader and vice versa.

17.2 Arbitration restrictions

Formatted: Bullets and Numbering

Here are the Sequencer arbitration restrictions:

Formatted: Bullets and Numbering

1) Cannot execute a serialized thread if the corresponding texture pending bit is set
2) Cannot allocate position if any older thread has not allocated position
3) If last thread is marked as not valid AND marked as last and we are about to execute the second to oldest thread also marked last then:
    a. Both threads must be from the same context (cannot allow a first thread)
    b. Must turn off the predicate optimization for the second thread
4) Cannot execute a texture clause if texture reads are pending
5) Cannot execute last if texture pending (even if not serial)

~~18. Vertex position exporting~~

Formatted: Bullets and Numbering

~~On clause 3 the vertex shader can export to the PA both the vertex position and the point sprite. It can also do so at clause 7 if not done at clause 3. The storage needed to perform the position export is at least 64x128 memories for the position and 64x32 memories for the sprite size. It is going to be taken in the pixel output fifo from the SX blocks. The clause where the position export occurs is specified by the EXPORT_LATE register. If turned on, it means that the export is going to occur at ALU clause 7 if unset position export occurs at clause 3.~~

~~19. Exporting Arbitration~~

Formatted: Bullets and Numbering

~~Here are the rules for co-issuing exporting ALU clauses.~~

~~1)Position exports and position exports cannot be co-issued.~~

Formatted: Bullets and Numbering

~~All other types of exports can be co-issued as long as there is place in the receiving buffer.~~

~~20. Exporting Rules~~

Formatted: Bullets and Numbering

~~20.1 Parameter caches exports~~

~~We support masking and out of order exports to the parameter caches. So one can export multiple times to the same PC line using different masks.~~

Formatted: Bullets and Numbering

~~20.2 Memory exports~~

~~Memory exports don't support masking. However, you can export out of order to memory locations.~~

Formatted: Bullets and Numbering

~~20.3 Position exports~~

~~Position exports have to be done IN ORDER and don't support masking.~~

Formatted: Bullets and Numbering

~~21.~~18. Export Types

The export type (or the location where the data should be put) is specified using the destination address field in the ALU instruction. Here is a list of all possible export modes:

Formatted: Bullets and Numbering

~~21.1~~18.1 Vertex Shading

0:15	- 16 parameter cache
16:31	- Empty (Reserved?)
32	- Export Address
33:40	- 8 vertex exports to the frame buffer and index
41:47	- Empty
48:55	- 8 debug export (interpret as normal vertex export)
60	- export addressing mode
61	- Empty
62	- position
63	- sprite size export that goes with position export (point_h,point_w,edgeflag,misc)

Formatted: Bullets and Numbering

~~21.2~~18.2 Pixel Shading

0	- Color for buffer 0 (primary)
1	- Color for buffer 1
2	- Color for buffer 2
3	- Color for buffer 3
4:7	- Empty
8	- Buffer 0 Color/Fog (primary)
9	- Buffer 1 Color/Fog
10	- Buffer 2 Color/Fog
11	- Buffer 3 Color/Fog
12:15	- Empty
16:31	- Empty (Reserved?)
32	- Export Address
33:40	- 8 exports for multipass pixel shaders.
41:47	- Empty
48:55	- 8 debug exports (interpret as normal pixel export)
60	- export addressing mode
61:62	- Empty
63	- Z for primary buffer (Z exported to 'alpha' component)

~~22.~~19. Special Interpolation modes

Formatted: Bullets and Numbering

~~22.1~~19.1 Real time commands

We are unable to use the parameter memory since there is no way for a command stream to write into it. Instead we need to add three 16x128 memories (one for each of three vertices x 16 interpolants). These will be mapped onto the register bus and written by type 0 packets, and output to the the parameter busses (the sequencer and/or PA need to be able to address the reatime parameter memory as well as the regular parameter store. For higher performance we should be able able to view them as two banks of 16 and do double buffering allowing one to be loaded, while the other is rasterized with. Most overlay shaders will need 2 or 4 scalar coordinates, one option might be to restrict the memory to 16x64 or 32x64 allowing only two interpolated scalars per cycle, the only problem I see with this is, if we view support for 16 vector-4 interpolants important (true only if we map Microsoft's high priority stream to the realtime stream), then the PA/sequencer need to support a realtime-specific mode where we need to address 32 vectors of parameters instead of 16. This mode is triggered by the primitive type: REAL TIME. The actual memories are in the in the SX blocks. The parameter data memories are hooked on the RBBM bus and are loaded by the CP using register mapped memory.

~~22.2~~19.2 Sprites/ XY screen coordinates/ FB information

Formatted: Bullets and Numbering

When working with sprites, one may want to overwrite the parameter 0 with SC generated data. Also, XY screen coordinates may be needed in the shader program. This functionality is controlled by the gen_I0 register (in SQ) in conjunction with the SND_XY register (in SC). Also it is possible to send the faceness information (for OGL front/back special operations) to the shader using the same control register. Here is a list of all the modes and how they interact together:

Gen_st is a bit taken from the interface between the SC and the SQ. This is the MSB of the primitive type. If the bit is set, it means we are dealing with Point AA, Line AA or sprite and in this case the vertex values are going to generated between 0 and 1.

Param_Gen_I0 disable, snd_xy disable, no gen_st – I0 = No modification
Param_Gen_I0 disable, snd_xy disable, gen_st – I0 = No modification
Param_Gen_I0 disable, snd_xy enable, no gen_st – I0 = No modification
Param_Gen_I0 disable, snd_xy enable, gen_st – I0 = No modification
Param_Gen_I0 enable, snd_xy disable, no gen_st – I0 = garbage, garbage, garbage, faceness
Param_Gen_I0 enable, snd_xy disable, gen_st – I0 = garbage, garbage, s, t
Param_Gen_I0 enable, snd_xy enable, no gen_st – I0 = screen x, screen y, garbage, faceness
Param_Gen_I0 enable, snd_xy enable, gen_st – I0 = screen x, screen y, s, t

~~22.3~~19.3 Auto generated counters

Formatted: Bullets and Numbering

In the cases we are dealing with multipass shaders, the sequencer is going to generate a vector count to be able to both use this count to write the 1st pass data to memory and then use the count to retrieve the data on the 2nd pass. The count is always generated in the same way but it is passed to the shader in a slightly different way depending on the shader type (pixel or vertex). This is toggled on and off using the GEN_INDEX register. The sequencer is going to keep two counters, one for pixels and one for vertices. Every time a full vector of vertices or pixels is written to the GPRs the counter is incremented. Every time a state change is detected, the corresponding counter is reset. While there is only one count broadcast to the GPRs, the LSB are hardwired to specific values making the index different for all elements in the vector.

~~22.3.1~~19.3.1 *Vertex shaders*

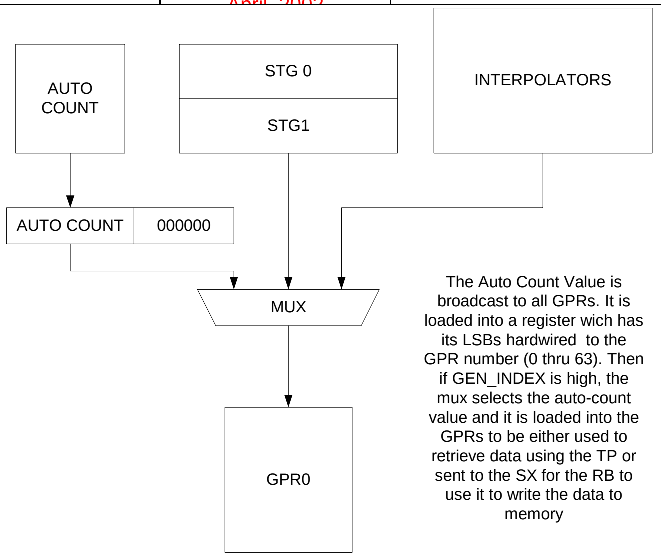
Formatted: Bullets and Numbering

In the case of vertex shaders, if GEN_INDEX is set, the data will be put into the x field of the third register (it means that the compiler must allocate 3 GPRs in all multipass vertex shader modes).

~~22.3.2~~19.3.2 *Pixel shaders*

Formatted: Bullets and Numbering

In the case of pixel shaders, if GEN_INDEX is set and Param_Gen_I0 is enabled, the data will be put in the x field of the 2nd register (R1.x), else if GEN_INDEX is set the data will be put into the x field of the 1st register (R0.x).



The Auto Count Value is broadcast to all GPRs. It is loaded into a register wich has its LSBs hardwired to the GPR number (0 thru 63). Then if GEN_INDEX is high, the mux selects the auto-count value and it is loaded into the GPRs to be either used to retrieve data using the TP or sent to the SX for the RB to use it to write the data to memory

Figure 12~~13~~: GPR input mux Control

Formatted: Bullets and Numbering

~~23.~~20. State management

Every clock, the sequencer will report to the CP the oldest states still in the pipe. These are the states of the programs as they enter the last ALU clause.

Formatted: Bullets and Numbering

~~23.1~~20.1 Parameter cache synchronization

In order for the sequencer not to begin a group of pixels before the associated group of vertices has finished, the sequencer will keep a 6 bit count per state (for a total of 8 counters). These counters are initialized to 0 and every time a vertex shader exports its data TO THE PARAMETER CACHE, the corresponding pointer is incremented. When the SC sends a new vector of pixels with the SC_SQ_new_vector bit asserted, the sequencer will first check if the count is greater than 0 before accepting the transmission (it will in fact accept the transmission but then lower its ready to receive). Then the sequencer waits for the count to go to one and decrements it. The sequencer can then issue the group of pixels to the interpolators. Every time the state changes, the new state counter is initialized to 0.

Formatted: Bullets and Numbering

~~24.~~21. XY Address imports

The SC will be able to send the XY addresses to the GPRs. It does so by interleaving the writes of the IJs (to the IJ buffer) with XY writes (to the XY buffer). Then when writing the data to the GPRs, the sequencer is going to interpolate the IJ data or pass the XY data thru a Fix→float converter and expander and write the converted values to the GPRs. The Xys are currently SCREEN SPACE COORDINATES. The values in the XY buffers will wrap. See section 19.2~~22.2~~ for details on how to control the interpolation in this mode.

Formatted: Bullets and Numbering

~~24.1~~21.1 Vertex indexes imports

In order to import vertex indexes, we have 16 8x96 staging registers. These are loaded one line at a time by the VGT block (96 bits). They are loaded in floating point format and can be transferred in 4 or 8 clocks to the GPRs.

~~25.~~22. Registers

Formatted: Bullets and Numbering

~~25.1~~22.1 Control

REG_DYNAMIC	Dynamic allocation (pixel/vertex) of the register file on or off.
REG_SIZE_PIX	Size of the register file's pixel portion (minimal size when dynamic allocation turned on)
REG_SIZE_VTX	Size of the register file's vertex portion (minimal size when dynamic allocation turned on)
ARBITRATION_POLICY	policy of the arbitration between vertexes and pixels
INST_BASE_VTX	start point for the vertex instruction store (RT always ends at vertex_base and Begins at 0)
INST_BASE_PIX	start point for the pixel shader instruction store
ONE_THREAD	debug state register. Only allows one program at a time into the GPRs
ONE_ALU	debug state register. Only allows one ALU program at a time to be executed (instead of 2)
INSTRUCTION	This is where the CP puts the base address of the instruction writes and type (auto-incremented on reads/writes) Register mapped
CONSTANTS	512*4 ALU constants + 32*6 Texture state 32 bits registers (logically mapped)
CONSTANTS_RT	256*4 ALU constants + 32*6 texture states? (physically mapped)
CONSTANT_EO_RT	This is the size of the space reserved for real time in the constant store (from 0 to CONSTANT_EO_RT). The re-mapping table operates on the rest of the memory
TSTATE_EO_RT	This is the size of the space reserved for real time in the fetch state store (from 0 to TSTATE_EO_RT). The re-mapping table operates on the rest of the memory

~~EXPORT_LATEControls whether or not we are exporting position from clause 3. If set, position exports occur at clause 7.~~

Formatted: Bullets and Numbering

~~25.2~~22.2 Context

~~VS_FETCH_{0…7}~~	~~eight 8 bit pointers to the location where each clauses control program is located~~
~~VS_ALU_{0…7}~~	~~eight 8 bit pointers to the location where each clauses control program is located~~
~~PS_FETCH_{0…7}~~	~~eight 8 bit pointers to the location where each clauses control program is located~~
~~PS_ALU_{0…7}~~	~~eight 8 bit pointers to the location where each clauses control program is located~~
PS_BASE	base pointer for the pixel shader in the instruction store
VS_BASE	base pointer for the vertex shader in the instruction store
VS_CF_SIZE	size of the vertex shader (# of instructions in control program/2)
PS_CF_SIZE	size of the pixel shader (# of instructions in control program/2)
PS_SIZE	size of the pixel shader (cntl+instructions)
VS_SIZE	size of the vertex shader (cntl+instructions)
PS_NUM_REG	number of GPRs to allocate for pixel shader programs
VS_NUM_REG	number of GPRs to allocate for vertex shader programs
PARAM_SHADE	One 16 bit register specifying which parameters are to be gouraud shaded (0 = flat, 1 = gouraud)
~~PROVO_VERT~~	~~0 : vertex 0, 1: vertex 1, 2: vertex 2, 3: Last vertex of the primitive~~
PARAM_WRAP	64 bits: for which parameters (and channels (xyzw)) do we do the cyl wrapping (0=linear, 1=cylindrical).
PS_EXPORT_MODE	0xxxx : Normal mode 1xxxx : Multipass mode If normal, bbbz where bbb is how many colors (0-4) and z is export z or not If multipass 1-12 exports for color.
VS_EXPORT_MODE	0: position (1 vector), 1: position (2 vectors), 3:multipass
VS_EXPORT_COUNT	Number of locations exported by the VS (and thus number of interpolated parameters) ~~{0…6}~~ ~~Six 4 bit counters representing the # of interpolated parameters exported in clause 7 (located in VS_EXPORT_COUNT_6) OR # of exported vectors to memory per clause in multipass mode (per clause)~~
PARAM_GEN_I0	Do we overwrite or not the parameter 0 with XY data and generated T and S values

GEN_INDEX	Auto generates an address from 0 to XX. Puts the results into R0-1 for pixel shaders and R2 for vertex shaders
CONST_BASE_VTX (9 bits)	Logical Base address for the constants of the Vertex shader
CONST_BASE_PIX (9 bits)	Logical Base address for the constants of the Pixel shader
CONST_SIZE_PIX (8 bits)	Size of the logical constant store for pixel shaders
CONST_SIZE_VTX (8 bits)	Size of the logical constant store for vertex shaders
INST_PRED_OPTIMIZE	Turns on the predicate bit optimization (if of, conditional_execute_predicates is always executed).
CF_BOOLEANS	256 boolean bits
CF_LOOP_COUNT	32x8 bit counters (number of times we traverse the loop)
CF_LOOP_START	32x8 bit counters (init value used in index computation)
CF_LOOP_STEP	32x8 bit counters (step value used in index computation)

Formatted: Bullets and Numbering

~~26.~~23. DEBUG Registers

~~26.1~~23.1 Context

DB_PROB_ADDR	instruction address where the first problem occurred
DB_PROB_COUNT	number of problems encountered during the execution of the program
DB_PROB_BREAK	break the clause if an error is found.
DB_ON	turns on an off debug method 2
DB_INST_COUNT	instruction counter for debug method 2
DB_BREAK_ADDR	break address for method number 2
~~DB_CLAUSE_MODE_ALU_{0…7}~~	~~clause mode for debug method 2 (0: normal, 1: addr, 2: kill)~~
~~DB_CLAUSE_MODE_FETCH_{0…7}~~	~~clause mode for debug method 2 (0: normal, 1: addr, 2: kill)~~

Formatted: Bullets and Numbering

~~26.2~~23.2 Control

DB_ALUCST_MEMSIZE	Size of the physical ALU constant memory
DB_TSTATE_MEMSIZE	Size of the physical texture state memory

Formatted: Bullets and Numbering

~~27.~~24. Interfaces

~~27.1~~24.1 External Interfaces

Whenever an x is used, it means that the bus is broadcast to all units of the same name. For example, if a bus is named SQ→SPx it means that SQ is going to broadcast the same information to all SP instances.

Formatted: Bullets and Numbering

~~27.2~~24.2 SC to SP Interfaces

Formatted: Bullets and Numbering

~~27.2.1~~24.2.1 *SC_SP#*

There is one of these interfaces at front of each of the SP (buffer to stage pixel interpolators). This interface transmits the I,J data for pixel interpolation. For the entire system, two quads per clock are transferred to the 4 SPs, so each of these 4 interfaces transmits one half of a quad per clock. The interface below describes a half of a quad worth of data.

The actual data which is transferred per quad is

Ref Pix I => S4.20 Floating Point I value
Ref Pix J => S4.20 Floating Point J value
Delta Pix I (x3) => S4.8 Floating Point Delta I value
Delta Pix J (x3) => S4.8 Floating Point Delta J value

This equates to a total of 128 bits which transferred over 2 clocks
and therefor needs an interface 64 bits wide

Additionally, X,Y data (12-bit unsigned fixed) is conditionally sent across this data bus over the same wires in an additional clock. The X,Y data is sent on the lower 24 bits of the data bus with faceness in the msb.
Transfers across these interfaces are synchronized with the SC_SQ IJ Control Bus transfers.

The data transfer across each of these busses is controlled by a IJ_BUF_INUSE_COUNT in the SC. Each time the SC has sent a pixel vector's worth of data to the SPs, he will increment the IJ_BUF_INUSE_COUNT count. Prior to sending the next pixel vectors data, he will check to make sure the count is less than MAX_BUFER_MINUS_2, if not the SC will stall until the SQ returns a pipelined pulse to decrement the count when he has scheduled a buffer free. Note: We could/may optimize for the case of only sending only IJ to use all the buffers to pre-load more. Currently it is planned for the SP to hold 2 double buffers of I,J data and two buffers of X,Y data, so if either X,Y or Centers and Centroids are on, then the SC can send two Buffers.

In at least the initial version, the SC shall send 16 quads per pixel vector even if the vector is not full. This will increment buffer write address pointers correctly all the time. (We may revisit this for both the SX,SP,SQ and add a EndOfVector signal on all interfaces to quit early. We opted for the simple mode first with a belief that only the end of packet and multiple new vector signals should cause a partial vector and that this would not really be significant performance hit.)

Name	Bits	Description
SC_SP#_data	64	IJ information sent over 2 clocks (or X,Y in 24 LSBs with faceness in upper bit) **Type 0 or 1**, First clock I, second clk J Field ULC URC LLC LRC Bits [63:39] [38:26] [25:13] [12:0] Format SE4M20 SE4M8 SE4M8 SE4M8 **Type 2** Field Face X Y Bits [63] [23:12] [11:0] Format Bit Unsigned Unsigned
SC_SP#_valid	1	Valid
SC_SP#_last_quad_data	1	This bit will be set on the last transfer of data per quad.
SC_SP#_type	2	0 -> Indicates centroids 1 -> Indicates centers 2 -> Indicates X,Y Data and faceness on data bus The SC shall look at state data to determine how many types to send for the interpolation process.

The # is included for clarity in the spec and will be replaced with a prefix of u#_ in the verilog module statement for the SC and the SP block will have neither because the instantiation will insert the prefix.

~~27.2.2~~24.2.2 SC_SQ

Formatted: Bullets and Numbering

This is the control information sent to the sequencer in order to synchronize and control the interpolation and/or loading data into the GPRs needed to execute a shader program on the sent pixels. This data will be sent over two clocks per transfer with 1 to 16 transfers. Therefore the bus (approx 92 bits) could be folded in half to approx 47 bits.

Name	Bits	Description
SC_SQ_data	46	Control Data sent to the SQ 1 clk transfers Event – valid data consist of event_id and state_id. Instruct SQ to post an event vector to send state id and event_id through request fifo and onto the reservation stations making sure state id and/or event_id gets back to the CP. Events only follow end of packets so no pixel vectors will be in progress.

		Empty Quad Mask – Transfer Control data consisting of pc_dealloc or new_vector. Receipt of this is to transfer pc_dealloc or new_vector without any valid quad data. New vector will always be posted to request fifo and pc_dealloc will be attached to any pixel vector outstanding or posted in request fifo if no valid quad outstanding. 2 clk transfers Quad Data Valid – Sending quad data with or without new_vector or pc_dealloc. New vector will be posted to request fifo with or without a pixel vector and pc_dealloc will be posted with a pixel vector unless none is in progress. In this case the pc_dealloc will be posted in the request queue. Filler quads will be transferred with The Quad mask set but the pixel corresponding pixel mask set to zero.
SC_SQ_valid	1	SC sending valid data, 2nd clk could be all zeroes

SC_SQ_data – first clock and second clock transfers are shown in the table below.

Name	BitField	Bits	Description
1st Clock Transfer			
SC_SQ_event	0	1	This transfer is a 1 clock event vector Force quad_mask = new_vector=pc_dealloc=0
SC_SQ_event_id	[2:1]	2	This field identifies the event 0 => denotes an End Of State Event 1 => TBD
SC_SQ_pc_dealloc	[5:3]	3	Deallocation token for the Parameter Cache
SC_SQ_new_vector	6	1	The SQ must wait for Vertex shader done count > 0 and after dispatching the Pixel Vector the SQ will decrement the count.
SC_SQ_quad_mask	[10:7]	4	Quad Write mask left to right SP0 => SP3
SC_SQ_end_of_prim	11	1	End Of the primitive
SC_SQ_state_id	[14:12]	3	State/constant pointer (6*3+3)
SC_SQ_pix_mask	[30:15]	16	Valid bits for all pixels SP0=>SP3 (UL,UR,LL,LR)
SC_SQ_prim_type	[33:31]	3	Stippled line and Real time command need to load tex cords from alternate buffer 000: Normal 010: Realtime 101: Line AA 110: Point AA (Sprite)
SC_SQ_provok_vtx	[35:34]	2	Provoking vertex for flat shading
SC_SQ_pc_ptr0	[46:36]	11	Parameter Cache pointer for vertex 0
2nd Clock Transfer			
SC_SQ_pc_ptr1	[10:0]	11	Parameter Cache pointer for vertex 1
SC_SQ_pc_ptr2	[21:11]	11	Parameter Cache pointer for vertex 2
SC_SQ_lod_correct	[45:22]	24	LOD correction per quad (6 bits per quad)

Name	Bits	Description

SQ_SC_free_buff	1	Pipelined bit that instructs SC to decrement count of buffers in use.
SQ_SC_dec_cntr_cnt	1	Pipelined bit that instructs SC to decrement count of new vector and/or event sent to prevent SC from overflowing SQ interpolator/Reservation request fifo.

The scan converter will submit a partial vector whenever:

1.) He gets a primitive marked with an end of packet signal.
2.) A current pixel vector is being assembled with at least one or more valid quads and the vector has been marked for deallocate when a primitive marked new_vector arrives. The Scan Converter will submit a partial vector (up to 16quads with zero pixel mask to fill out the vector) prior to submitting the new_vector marker\primitive.

(This will prevent a hang which can be demonstrated when all primitives in a packet three vectors are culled except for a one quad primitive that gets marked pc_dealloc (vertices maximum size). In this case two new_vectors are submitted and processed, but then one valid quad with the pc_dealloc creates a vector and then the new would wait for another vertex vector to be processed, but the one being waited for could never export until the pc_dealloc signal made it through and thus the hang.)

~~27.2.3~~24.2.3 SQ to SX: Interpolator bus

Formatted: Bullets and Numbering

Name	**Direction**	**Bits**	**Description**
SQ_SXx_interp_flat_vtx	SQ→SPx	2	Provoking vertex for flat shading
SQ_SXx_interp_flat_gouraud	SQ→SPx	1	Flat or gouraud shading
SQ_SXx_interp_cyl_wrap	SQ→SPx	4	Wich channel needs to be cylindrical wrapped
SQ_SXx_pc_ptr0	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_pc_ptr1	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_pc_ptr2	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_rt_sel	SQ→SXx	1	Selects between RT and Normal data
SQ_SXx_pc_wr_en	SQ→SXx	1	Write enable for the PC memories
SQ_SXx_pc_wr_addr	SQ→SXx	7	Write address for the PCs
SQ_SXx_pc_channel_mask	SQ→SXx	4	Channel mask

~~27.2.4~~24.2.4 SQ to SP: Staging Register Data

Formatted: Bullets and Numbering

This is a broadcast bus that sends the VSISR information to the staging registers of the shader pipes.

Name	**Direction**	**Bits**	**Description**
SQ_SPx_vsr_data	SQ→SPx	96	Pointers of indexes or HOS surface information
SQ_SPx_vsr_double	SQ→SPx	1	0: Normal 96 bits per vert 1: double 192 bits per vert
SQ_SP0_ vsr_valid	SQ→SP0	1	Data is valid
SQ_SP1_ vsr_ valid	SQ→SP1	1	Data is valid
SQ_SP2_ vsr_ valid	SQ→SP2	1	Data is valid
SQ_SP3_ vsr_ valid	SQ→SP3	1	Data is valid
SQ_SPx_vsr_read	SQ→SPx	1	Increment the read pointers

~~27.2.5~~24.2.5 VGT to SQ : Vertex interface

Formatted: Bullets and Numbering

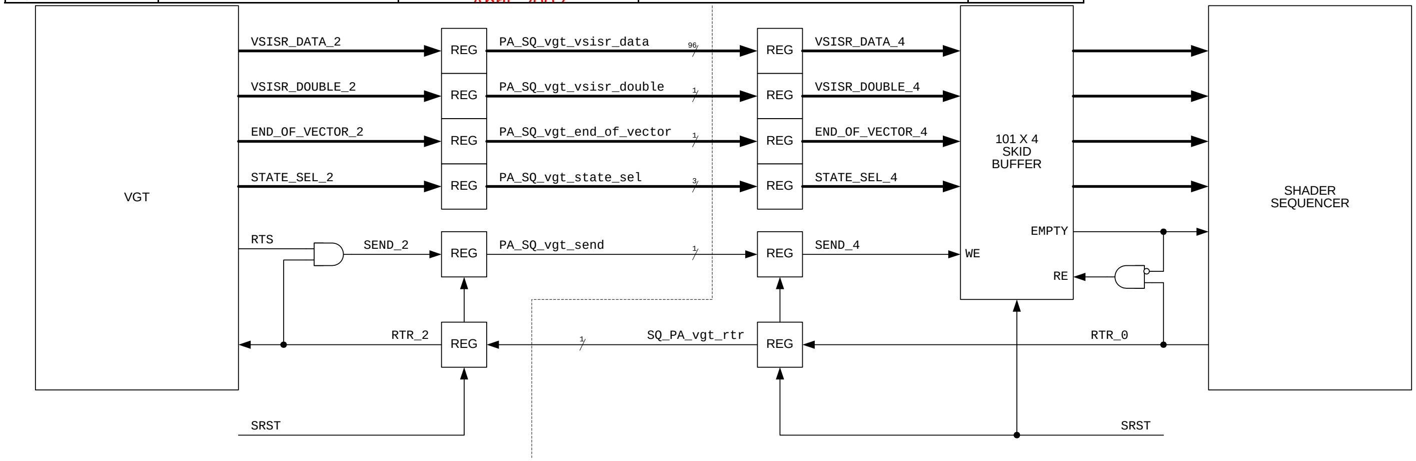
~~27.2.5.1~~24.2.5.1 Interface Signal Table

The area difference between the two methods is not sufficient to warrant complicating the interface or the state requirements of the VSISRs. **Therefore, the POR for this interface is that the VGT will transmit the data to the VSISRs (via the Shader Sequencer) in full, 32-bit floating-point format.** The VGT can transmit up to six 32-bit floating-point values to each VSISR where four or more values require two transmission clocks. The data bus is 96 bits wide.

Name	Bits	Description
VGT_SQ_vsisr_data	96	Pointers of indexes or HOS surface information
VGT_SQ_vsisr_double	1	0: Normal 96 bits per vert 1: double 192 bits per vert
VGT_SQ_end_of_vector	1	Indicates the last VSISR data set for the current process vector (for double vector data, "end_of_vector" is set on the first vector)
VGT_SQ_indx_valid	1	Vsisr data is valid
VGT_SQ_state	3	Render State (6*3+3 for constants). This signal is guaranteed to be correct when "VGT_SQ_vgt_end_of_vector" is high.
VGT_SQ_send	1	Data on the VGT_SQ is valid receive (see write-up for standard R400 SEND/RTR interface handshaking)
SQ_VGT_rtr	1	Ready to receive (see write-up for standard R400 SEND/RTR interface handshaking)

~~27.2.5.2~~24.2.5.2 Interface Diagrams

Formatted: Bullets and Numbering



VGT
VSISR_DATA_2
VSISR_DOUBLE_2
END_OF_VECTOR_2
STATE_SEL_2
RTS
SEND_2
RTR_2
SRST
REG
PA_SQ_vgt_vsisr_data
96
PA_SQ_vgt_vsisr_double
1
PA_SQ_vgt_end_of_vector
1
PA_SQ_vgt_state_sel
3
PA_SQ_vgt_send
1
SQ_PA_vgt_rtr
1
VSISR_DATA_4
VSISR_DOUBLE_4
END_OF_VECTOR_4
STATE_SEL_4
SEND_4
101 X 4
SKID
BUFFER
EMPTY
WE
RE
RTR_0
SRST
SHADER
SEQUENCER

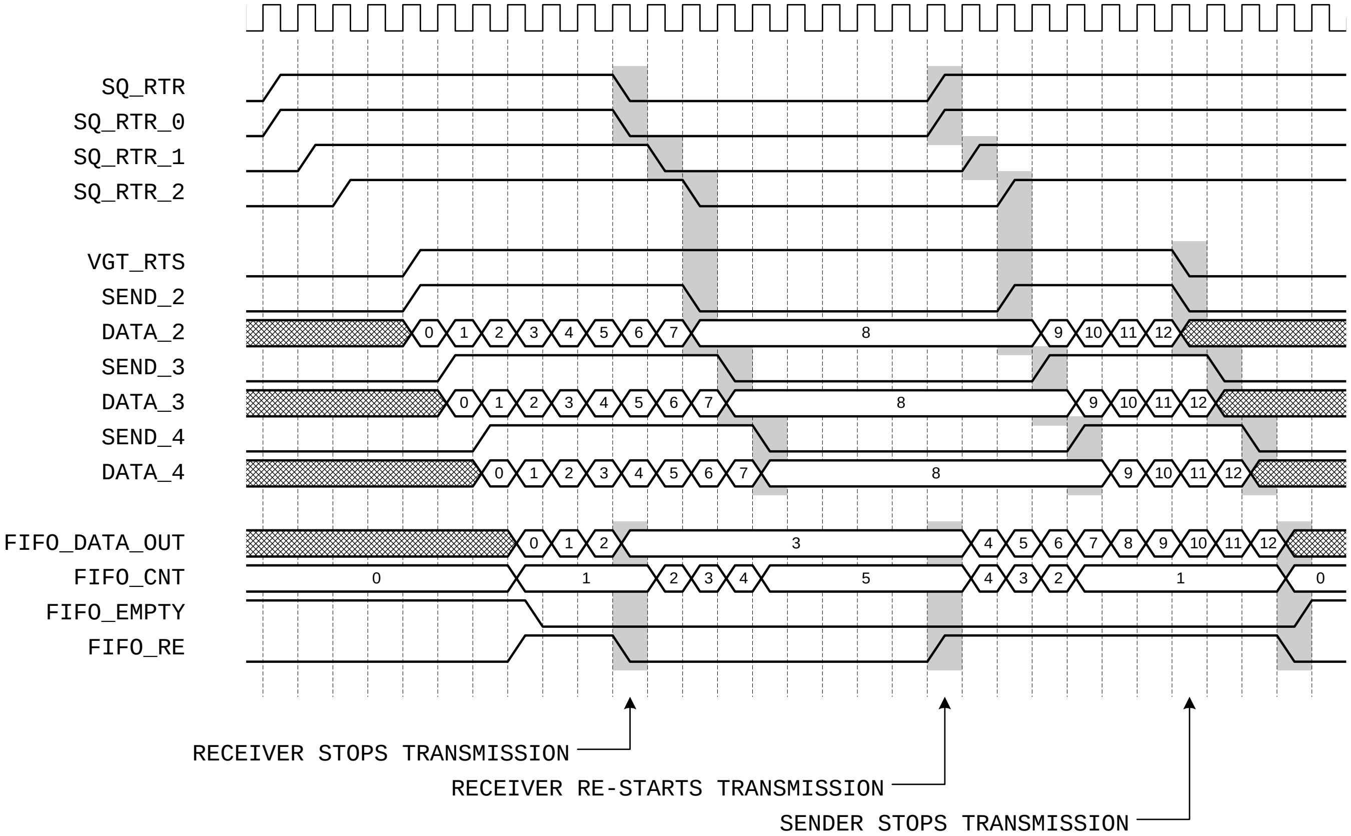


Figure 1. Detailed Logical Diagram for PA_SQ_vgt Interface.

~~27.2.6~~24.2.6 *SQ to SX: Control bus*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SXx_exp_type	SQ→SXx	2	00: Pixel without z (1 to 4 buffers) 01: Pixel with z (1 to 4 buffers) 10: Position (1 or 2 results) 11: Pass thru (4,8 or 12 results aligned)
SQ_SXx_exp_number	SQ→SXx	2	Number of locations needed in the export buffer (encoding depends on the type see bellow).
SQ_SXx_exp_alu_id	SQ→SXx	1	ALU ID
SQ_SXx_exp_valid	SQ→SXx	1	Valid bit
SQ_SXx_exp_state	SQ→SXx	3	State Context
SQ_SXx_free_done	SQ→SXx	1	Pulse to indicate that the previous export is finished (this can be sent with or without the other fields of the interface)
SQ_SXx_free_alu_id	SQ→SXx	1	ALU ID

Formatted

Formatted

Formatted

Formatted

Formatted

Formatted

Formatted

Depending on the type the number of export location changes:

Formatted: Bullets and Numbering

- Type 00 : Pixels without Z
  - o 00 = 1 buffer
  - o 01 = 2 buffers
  - o 10 = 3 buffers
  - o 11 = 4 buffer
- Type 01: Pixels with Z
  - o 00 = 2 Buffers (color + Z)
  - o 01 = 3 buffers (2 color + Z)
  - o 10 = 4 buffers (3 color + Z)
  - o 11 = 5 buffers (4 color + Z)
- Type 10 : Position export
  - o 00 = 1 position
  - o 01 = 2 positions
  - o 1X = Undefined
- Type 11: Pass Thru
  - o 00 = 4 buffers
  - o 01 = 8 buffers
  - o 10 = 12 buffers
  - o 11 = Undefined

Below the thick black line is the end of transfer packet that tells the SX that a given export is finished. The report packet **will always arrive either before or at the same time than the next export to the same ALU id**.~~These fields are sent every time the sequencer picks an exporting clause for execution.~~

~~27.2.7~~24.2.7 *SX to SQ : Output file control*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SXx_SQ_exp_count_rdy	SXx→SQ	1	Raised by SX0 to indicate that the following two fields reflect the result of the most recent export
SXx_SQ_exp_pos_avail	SXx→SQ	1	Specifies whether there is room for another position.
SXx_SQ_exp_buf_avail	SXx→SQ	7	Specifies the space available in the output buffers. 0: buffers are full 1: 2K-bits available (32-bits for each of the 64 pixels in a clause) ... 64: 128K-bits available (16 128-bit entries for each of 64 pixels) 65-127: RESERVED

~~27.2.8~~24.2.8 *SQ to TP: Control bus*

Formatted: Bullets and Numbering

Once every clock, the fetch unit sends to the sequencer on which ~~clause~~ RS line it is now working and if the data in the GPRs is ready or not. This way the sequencer can update the fetch valid bits ~~counters~~ flags for the reservation station ~~fifos~~. The sequencer also provides the instruction and constants for the fetch to execute and the address in the register file where to write the fetch return data.

Name ~~**Name**~~	**Direction**~~**Direction**~~	**Bits**~~**Bits**~~	**Description**~~**Description**~~
TPx_SQ_data_rdy~~TPx_SQ_data_rdy~~	TPx→ SQ~~TPx→ SQ~~	1~~1~~	Data ready~~Data ready~~
TPx_SQ_rs_line_num~~TPx_SQ_clause_num~~	TPx→ SQ~~TPx→ SQ~~	6~~3~~	Line number in the Reservation station~~Clause number~~
TPx_SQ_type~~TPx_SQ_type~~	TPx→ SQ~~TPx→ SQ~~	1~~1~~	Type of data sent (0:PIXEL, 1:VERTEX)~~Type of data sent (0:PIXEL, 1:VERTEX)~~
SQ_TPx_send~~SQ_TPx_send~~	SQ→TPx~~SQ→TPx~~	1~~1~~	Sending valid data~~Sending valid data~~
SQ_TPx_const~~SQ_TPx_const~~	SQ→TPx~~SQ→TPx~~	48~~48~~	Fetch state sent over 4 clocks (192 bits total)~~Fetch state sent over 4 clocks (192 bits total)~~
SQ_TPx_instr~~SQ_TPx_instr~~	SQ→TPx~~SQ→TPx~~	24~~24~~	Fetch instruction sent over 4 clocks~~Fetch instruction sent over 4 clocks~~
SQ_TPx_end_of_group~~SQ_TPx_end_of_clause~~	SQ→TPx~~SQ→TPx~~	1~~1~~	Last instruction of the group~~Last instruction of the clause~~
SQ_TPx_Type~~SQ_TPx_Type~~	SQ→TPx~~SQ→TPx~~	1~~1~~	Type of data sent (0:PIXEL, 1:VERTEX)~~Type of data sent (0:PIXEL, 1:VERTEX)~~
SQ_TPx_gpr_phase~~SQ_TPx_gpr_phase~~	SQ→TPx~~SQ→TPx~~	2~~2~~	Write phase signal~~Write phase signal~~
SQ_TP0_lod_correct~~SQ_TP0_lod_correct~~	SQ→TP0~~SQ→TP0~~	6~~6~~	LOD correct 3 bits per comp 2 components per quad ~~LOD correct 3 bits per comp 2 components per quad~~
SQ_TP0_pix_mask~~SQ_TP0_pix_mask~~	SQ→TP0~~SQ→TP0~~	4~~4~~	Pixel mask 1 bit per pixel~~Pixel mask 1 bit per pixel~~
SQ_TP1_lod_correct~~SQ_TP1_lod_correct~~	SQ→TP1~~SQ→TP1~~	6~~6~~	LOD correct 3 bits per comp 2 components per quad ~~LOD correct 3 bits per comp 2 components per quad~~
SQ_TP1_pix_mask~~SQ_TP1_pix_mask~~	SQ→TP1~~SQ→TP1~~	4~~4~~	Pixel mask 1 bit per pixel~~Pixel mask 1 bit per pixel~~
SQ_TP2_lod_correct~~SQ_TP2_lod_correct~~	SQ→TP2~~SQ→TP2~~	6~~6~~	LOD correct 3 bits per comp 2 components per quad ~~LOD correct 3 bits per comp 2 components per quad~~
SQ_TP2_pix_mask~~SQ_TP2_pix_mask~~	SQ→TP2~~SQ→TP2~~	4~~4~~	Pixel mask 1 bit per pixel~~Pixel mask 1 bit per pixel~~
SQ_TP3_lod_correct~~SQ_TP3_lod_correct~~	SQ→TP3~~SQ→TP3~~	6~~6~~	LOD correct 3 bits per comp 2 components per quad ~~LOD correct 3 bits per comp 2 components per quad~~
SQ_TP3_pix_mask~~SQ_TP3_pix_mask~~	SQ→TP3~~SQ→TP3~~	4~~4~~	Pixel mask 1 bit per pixel~~Pixel mask 1 bit per pixel~~

Formatted

Formatted

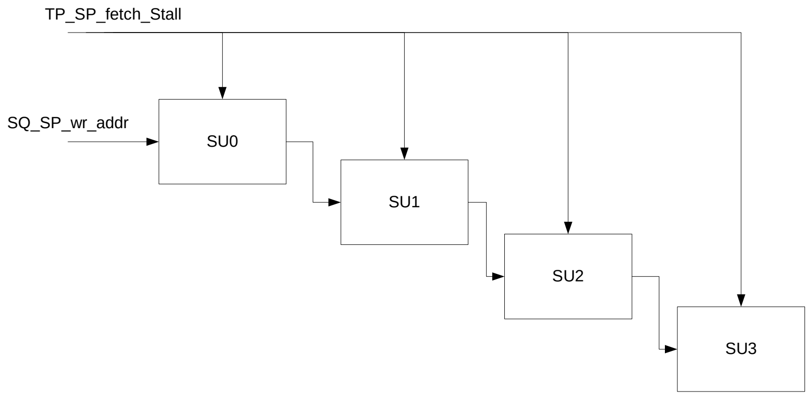
SQ_TPx_rs_line_num~~SQ_TPx_clause_num~~	SQ→TPx~~SQ→TPx~~	6~~3~~	Line number in the Reservation station~~Clause number~~
SQ_TPx_write_gpr_index~~SQ_TPx_write_gpr_index~~	SQ->TPx~~SQ->TPx~~	7~~7~~	Index into Register file for write of returned Fetch Data~~Index into Register file for write of returned Fetch Data~~

Formatted

Formatted: Bullets and Numbering

~~27.2.9~~24.2.9 *TP to SQ: Texture stall*

The TP sends this signal to the SQ and the SPs when its input buffer is full.



Name	Direction	Bits	Description
TP_SQ_fetch_stall	TP→ SQ	1	Do not send more texture request if asserted

Formatted: Bullets and Numbering

~~27.2.10~~24.2.10 *SQ to SP: Texture stall*

Name	Direction	Bits	Description
SQ_SPx_fetch_stall	SQ→SPx	1	Do not send more texture request if asserted

Formatted: Bullets and Numbering

~~27.2.11~~24.2.11 *SQ to SP: GPR and auto counter*

Name	Direction	Bits	Description
SQ_SPx_gpr_wr_addr	SQ→SPx	7	Write address
SQ_SPx_gpr_rd_addr	SQ→SPx	7	Read address
SQ_SPx_gpr_rd_en	SQ→SPx	1	Read Enable
SQ_SPx_gpr_wr_en	SQ→SPx	1	Write Enable for the GPRs
SQ_SPx_gpr_phase	SQ→SPx	2	The phase mux (arbitrates between inputs, ALU SRC reads and writes)
SQ_SPx_channel_mask	SQ→SPx	4	The channel mask
SQ_SPx_gpr_input_sel	SQ→SPx	2	When the phase mux selects the inputs this tells from which source to read from: Interpolated data, VTX0, VTX1, autogen counter.
SQ_SPx_auto_count	SQ→SPx	12?	Auto count generated by the SQ, common for all shader pipes

~~27.2.12~~24.2.12 SQ to SPx: Instructions

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SPx_instr_start	SQ→SPx	1	Instruction start
SQ_SP_instr	SQ→SPx	21	Transferred over 4 cycles 0: SRC A Select 2:0 SRC A Argument Modifier 3:3 SRC A swizzle 11:4 VectorDst 17:12 Unused 20:18 --- - 1: SRC B Select 2:0 SRC B Argument Modifier 3:3 SRC B swizzle 11:4 ScalarDst 17:12 Unused 20:18 --- - 2: SRC C Select 2:0 SRC C Argument Modifier 3:3 SRC C swizzle 11:4 Unused 20:12 --- - 3: Vector Opcode 4:0 Scalar Opcode 10:5 Vector Clamp 11:11 Scalar Clamp 12:12 Vector Write Mask 16:13 Scalar Write Mask 20:17
SQ_SPx_exp_alu_id	SQ→SPx	1	ALU ID
SQ_SPx_exporting	SQ→SPx	2	0: Not Exporting 1: Vector Exporting 2: Scalar Exporting
SQ_SPx_stall	SQ→SPx	1	Stall signal
SQ_SP0_write_mask	SQ→SP0	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP1_ write_mask	SQ→SP1	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP2_ write_mask	SQ→SP2	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP3_ write_mask	SQ→SP3	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock

~~27.2.13~~24.2.13 SP to SQ: Constant address load/ Predicate Set

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SP0_SQ_const_addr	SP0→SQ	36	Constant address load / predicate vector load (4 bits only) to the sequencer
SP0_SQ_valid	SP0→SQ	1	Data valid
SP1_SQ_const_addr	SP1→SQ	36	Constant address load / predicate vector load (4 bits only) to the sequencer

SP1_SQ_valid	SP1→SQ	1	Data valid
SP2_SQ_const_addr	SP2→SQ	36	Constant address load / predicate vector load (4 bits only) to the sequencer
SP2_SQ_valid	SP2→SQ	1	Data valid
SP3_SQ_const_addr	SP3→SQ	36	Constant address load / predicate vector load (4 bits only) to the sequencer
SP3_SQ_valid	SP3→SQ	1	Data valid

~~27.2.14~~24.2.14 *SQ to SPx: constant broadcast*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SPx_const	SQ→SPx	128	Constant broadcast

~~27.2.15~~24.2.15 *SP0 to SQ: Kill vector load*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SP0_SQ_kill_vect	SP0→SQ	4	Kill vector load
SP1_SQ_kill_vect	SP1→SQ	4	Kill vector load
SP2_SQ_kill_vect	SP2→SQ	4	Kill vector load
SP3_SQ_kill_vect	SP3→SQ	4	Kill vector load

~~27.2.16~~24.2.16 *SQ to CP: RBBM bus*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_RBB_rs	SQ→CP	1	Read Strobe
SQ_RBB_rd	SQ→CP	32	Read Data
SQ_RBBM_nrtrtr	SQ→CP	1	Optional
SQ_RBBM_rtr	SQ→CP	1	Real-Time (Optional)

~~27.2.17~~24.2.17 *CP to SQ: RBBM bus*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
rbbm_we	CP→SQ	1	Write Enable
rbbm_a	CP→SQ	15	Address -- Upper Extent is TBD (16:2)
rbbm_wd	CP→SQ	32	Data
rbbm_be	CP→SQ	4	Byte Enables
rbbm_re	CP→SQ	1	Read Enable
rbb_rs0	CP→SQ	1	Read Return Strobe 0
rbb_rs1	CP→SQ	1	Read Return Strobe 1
rbb_rd0	CP→SQ	32	Read Data 0
rbb_rd1	CP→SQ	32	Read Data 0
RBBM_SQ_soft_reset	CP→SQ	1	Soft Reset

~~27.2.18~~24.2.18 *SQ to CP: State report*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_CP_vs_event	SQ→CP	1	Vertex Shader Event
SQ_CP_vs_eventid	SQ→CP	2	Vertex Shader Event ID
SQ_CP_ps_event	SQ→CP	1	Pixel Shader Event
SQ_CP_ps_eventid	SQ→CP	2	Pixel Shader Event ID

eventid = 0 => *sEndOfState (i.e. VsEndOfState)
eventid = 1 => *sDone (i.e. VsDone)

So, the CP will assume the Vs is done with a state whenever it gets a pulse on the SQ_CP_vs_event and the SQ_CP_vs_eventid = 0.

Formatted: Bullets and Numbering

24.3 Example of control flow program execution

We now provide some examples of execution to better illustrate the new design.

Given the program:

Alu 0
Alu 1
Tex 0
Tex 1
Alu 3 Serial
Alu 4
Tex 2
Alu 5
Alu 6 Serial
Tex 3
Alu 7
Alloc Position 1 buffer
Alu 8 Export
Tex 4
Alloc Parameter 3 buffers
Alu 9 Export 0
Tex 5
Alu 10 Serial Export 2
Alu 11 Export 1 End

Would be converted into the following CF instructions:

```
Execute Alu 0 Alu 0 Tex 0 Tex 0 Alu 1 Alu 0 Tex 0 Alu 0 Alu 1 Tex 0
Execute Alu 0
Alloc Position 1
Execute Alu 0 Tex 0
Alloc Param 3
Execute Alu 0 Tex 0 Alu 1 Alu 0 End
```

And the execution of this program would look like this:

Put thread in Vertex RS:

Control Flow Instruction Pointer (12 bits), (CFP)
Execution Count Marker (3 or 4 bits), (ECM)
Loop Iterators (4x9 bits), (LI)
Call return pointers (4x12 bits), (CRP)
Predicate Bits(4x64 bits), (PB)
Export ID (1 bit), (EXID)
GPR Base Ptr (8 bits), (GPR)
Export Base Ptr (7 bits), (EB)
Context Ptr (3 bits).(CPTR)
LOD correction bits (16x6 bits) (LOD)

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	0	0	0	0	0	0	0	0	0

Valid Thread (VALID)
Texture/ALU engine needed (TYPE)
Texture Reads are outstanding (PENDING)
Waiting on Texture Read to Complete (SERIAL)
Allocation Wait (2 bits) (ALLOC)

00 – No allocation needed
01 – Position export allocation needed (ordered export)
10 – Parameter or pixel export needed (ordered export)
11 – pass thru (out of order export)
Allocation Size (4 bits) (SIZE)
Position Allocated (POS_ALLOC)
First thread of a new context (FIRST)
Last (1 bit), (LAST)

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	0	0	0	0	0	1	0

Then the thread is picked up for the execution of the first control flow instruction:

```
Execute Alu 0 Alu 0 Tex 0 Tex 0 Alu 1 Alu 0 Tex 0 Alu 0 Alu 1 Tex 0
```

It executes the first two ALU instructions and goes back to the RS for a resource request change. Here is the state returned to the RS:

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	2	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	0	0	0	0	0	1	0

Then when the texture pipe frees up, the arbiter picks up the thread to issue the texture reads. The thread comes back in this state:

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	4	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	1	0	0	0	1	0

Because of the serial bit the arbiter must wait for the texture to return and clear the PENDING bit before it can pick the thread up. Lets say that the texture reads are complete, then the arbiter picks up the thread and returns it in this state:

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	6	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	0	0	0	0	0	1	0

Again the TP frees up, the arbiter picks up the thread and executes. It returns in this state:

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	7	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	0	0	0	1	0

Now, even if the texture has not returned we can still pick up the thread for ALU execution because the serial bit is not set. The thread will however come back to the RS for the second ALU instruction because it has the serial bit set.

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	8	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	1	0	0	0	1	0

As soon as the TP clears the pending bit the thread is picked up and returns:

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	9	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	0	0	0	0	0	1	0

Picked up by the TP and returns:

```
Execute Alu 0
```

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
1	0	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	0	0	0	1	0

Picked up by the ALU and returns (lets say the TP has not returned yet):

```
Alloc Position 1
```

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
2	0	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	01	1	0	1	0

If the SX has the place for the export, the SQ is going to allocate and pick up the thread for execution. It returns to the RS in this state:

```
Execute Alu 0 Tex 0
```

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
3	1	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	1	0	0	0	1	1	0

Now, since the TP has not returned yet, we must wait for it to return because we cannot issue multiple texture requests. The TP returns, clears the PENDING bit and we proceed:

```
Alloc Param 3
```

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
4	0	0	0	0	1	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	10	3	1	1	0

Once again the SQ makes sure the SX has enough room in the Parameter cache before it can pick up this thread.

```
Execute Alu 0 Tex 0 Alu 1 Alu 0 End
```

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
5	1	0	0	0	1	0	100	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	1	0	0	0	1	1	0

This executes on the TP and then returns:

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
5	2	0	0	0	1	0	100	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	1	0	0	1	1	1

Waits for the TP to return because of the textures reads are pending (and SERIAL in this case). Then executes and does not return to the RS because the LAST bit is set. This is the end of this thread and before dropping it on the floor, the SQ notifies the SX of export completion.

Formatted: Bullets and Numbering

~~28.~~25. Open issues

Need to do some testing on the size of the register file as well as on the register file allocation method (dynamic VS static).

Saving power?

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 2015~~2~~ May, 200219 April, 2002	GEN-CXXXXX-REVA	1 of 54

Author: Laurent Lefebvre

Issue To:	**Copy No:**

R400 Sequencer Specification

SQ

Version 2.01~~0~~

Overview: This is an architectural specification for the R400 Sequencer block (SEQ). It provides an overview of the required capabilities and expected uses of the block. It also describes the block interfaces, internal sub-blocks, and provides internal state diagrams.

AUTOMATICALLY UPDATED FIELDS:

Document Location:	C:\perforce\r400\doc_lib\design\blocks\sq\R400_Sequencer.doc
Current Intranet Search Title:	R400 Sequencer Specification

APPROVALS	
Name/Dept	Signature/Date

Remarks:

THIS DOCUMENT CONTAINS CONFIDENTIAL INFORMATION THAT COULD BE SUBSTANTIALLY DETRIMENTAL TO THE INTEREST OF ATI TECHNOLOGIES INC. THROUGH UNAUTHORIZED USE OR DISCLOSURE.

ATI 2029
LG v. ATI
IPR2015-00326

Table Of Contents

1. OVERVIEW ..... 9~~7~~
1.1 Top Level Block Diagram ..... 11~~9~~
1.2 Data Flow graph (SP) ..... 12~~10~~
1.3 Control Graph ..... 13~~11~~
2. INTERPOLATED DATA BUS ..... 13~~11~~
3. INSTRUCTION STORE ..... 16~~14~~
4. SEQUENCER INSTRUCTIONS ..... 16~~14~~
5. CONSTANT STORES ..... 16~~14~~
5.1 Memory organizations ..... 16~~14~~
5.2 Management of the Control Flow Constants ..... 17~~15~~
5.3 Management of the re-mapping tables ..... 17~~15~~
5.3.1 R400 Constant management ..... 17~~15~~
5.3.2 Proposal for R400LE constant management ..... 17~~15~~
5.3.3 Dirty bits ..... 19~~17~~
5.3.4 Free List Block ..... 19~~17~~
5.3.5 De-allocate Block ..... 20~~18~~
5.3.6 Operation of Incremental model ..... 20~~18~~
5.4 Constant Store Indexing ..... 20~~18~~
5.5 Real Time Commands ..... 21~~19~~
5.6 Constant Waterfalling ..... 21~~19~~
6. LOOPING AND BRANCHES ..... 22~~20~~
6.1 The controlling state. ..... 22~~20~~
6.2 The Control Flow Program ..... 22~~20~~
6.2.1 Control flow instructions table ..... 23~~21~~
6.3 Implementation ..... 24~~22~~
6.4 Data dependant predicate instructions ..... 26~~24~~
6.5 HW Detection of PV,PS ..... 27~~24~~
6.6 Register file indexing ..... 27~~24~~
6.7 Debugging the Shaders ..... 27~~25~~
6.7.1 Method 1: Debugging registers ..... 27~~25~~
6.7.2 Method 2: Exporting the values in the GPRs ..... 28~~25~~
7. PIXEL KILL MASK ..... 28~~26~~
8. MULTIPASS VERTEX SHADERS (HOS) ..... 28~~26~~
9. REGISTER FILE ALLOCATION ..... 28~~26~~
10. FETCH ARBITRATION ..... 29~~27~~
11. ALU ARBITRATION ..... 29~~27~~
12. HANDLING STALLS ..... 30~~28~~
13. CONTENT OF THE RESERVATION STATION FIFOS ..... 30~~28~~
14. THE OUTPUT FILE ..... 30~~28~~
15. IJ FORMAT ..... 30~~28~~
15.1 Interpolation of constant attributes ..... 31~~29~~
16. STAGING REGISTERS ..... 31~~29~~

17. THE PARAMETER CACHE **33~~30~~**
17.1 Export restrictions 34~~30~~
17.1.1 Pixel exports: 34~~30~~
17.1.2 Vertex exports: 34~~30~~
17.1.3 Pass thru exports: 34~~30~~
17.2 Arbitration restrictions 34~~30~~
18. EXPORT TYPES **34~~31~~**
18.1 Vertex Shading 34~~31~~
18.2 Pixel Shading 35~~31~~
19. SPECIAL INTERPOLATION MODES **35~~31~~**
19.1 Real time commands 35~~31~~
19.2 Sprites/ XY screen coordinates/ FB information 35~~32~~
19.3 Auto generated counters 36~~32~~
19.3.1 Vertex shaders 36~~32~~
19.3.2 Pixel shaders 36~~32~~
20. STATE MANAGEMENT **36~~33~~**
20.1 Parameter cache synchronization 36~~33~~
21. XY ADDRESS IMPORTS **37~~33~~**
21.1 Vertex indexes imports 37~~33~~
22. REGISTERS **37~~34~~**
22.1 Control 37~~34~~
22.2 Context 37~~34~~
23. DEBUG REGISTERS **38~~35~~**
23.1 Context 38~~35~~
23.2 Control 38~~35~~
24. INTERFACES **38~~35~~**
24.1 External Interfaces 38~~35~~
24.2 SC to SP Interfaces 38~~35~~
24.2.1 SC_SP# 38~~35~~
24.2.2 SC_SQ 39~~36~~
24.2.3 SQ to SX: Interpolator bus 41~~38~~
24.2.4 SQ to SP: Staging Register Data 41~~38~~
24.2.5 VGT to SQ : Vertex interface 41~~38~~
24.2.6 SQ to SX: Control bus 45~~41~~
24.2.7 SX to SQ : Output file control 45~~41~~
24.2.8 SQ to TP: Control bus 46~~42~~
24.2.9 TP to SQ: Texture stall 46~~42~~
24.2.10 SQ to SP: Texture stall 47~~42~~
24.2.11 SQ to SP: GPR and auto counter 47~~43~~
24.2.12 SQ to SPx: Instructions 48~~44~~
24.2.13 SP to SQ: Constant address load/ Predicate Set 48~~44~~
24.2.14 SQ to SPx: constant broadcast 49~~45~~

24.2.15 SP0 to SQ: Kill vector load .... 49~~45~~
24.2.16 SQ to CP: RBBM bus .... 49~~45~~
24.2.17 CP to SQ: RBBM bus .... 49~~45~~
24.2.18 SQ to CP: State report .... 49~~45~~
24.3 Example of control flow program execution .... 50~~45~~
25. OPEN ISSUES .... 54~~50~~
~~**1. OVERVIEW .... 6**~~
~~1.1 Top Level Block Diagram .... 8~~
~~1.2 Data Flow graph (SP) .... 9~~
~~1.3 Control Graph .... 10~~
~~**2. INTERPOLATED DATA BUS .... 10**~~
~~**3. INSTRUCTION STORE .... 13**~~
~~**4. SEQUENCER INSTRUCTIONS .... 13**~~
~~**5. CONSTANT STORES .... 13**~~
~~5.1 Memory organizations .... 13~~
~~5.2 Management of the Control Flow Constants .... 14~~
~~5.3 Management of the re-mapping tables .... 14~~
~~5.3.1 R400 Constant management .... 14~~
~~5.3.2 Proposal for R400LE constant management .... 14~~
~~5.3.3 Dirty bits .... 16~~
~~5.3.4 Free List Block .... 16~~
~~5.3.5 De-allocate Block .... 17~~
~~5.3.6 Operation of Incremental model .... 17~~
~~5.4 Constant Store Indexing .... 17~~
~~5.5 Real Time Commands .... 18~~
~~5.6 Constant Waterfalling .... 18~~
~~**6. LOOPING AND BRANCHES .... 19**~~
~~6.1 The controlling state. .... 19~~
~~6.2 The Control Flow Program .... 19~~
~~6.3 Data dependant predicate instructions .... 23~~
~~6.4 HW Detection of PV,PS .... 23~~
~~6.5 Register file indexing .... 23~~
~~6.6 Predicated Instruction support for Texture clauses .... 24~~
~~6.7 Debugging the Shaders .... 24~~
~~6.7.1 Method 1: Debugging registers .... 24~~
~~6.7.2 Method 2: Exporting the values in the GPRs (12) .... 24~~
~~**7. PIXEL KILL MASK .... 25**~~
~~**8. MULTIPASS VERTEX SHADERS (HOS) .... 25**~~
~~**9. REGISTER FILE ALLOCATION .... 25**~~
~~**10. FETCH ARBITRATION .... 26**~~
~~**11. ALU ARBITRATION .... 26**~~
~~**12. HANDLING STALLS .... 27**~~
~~**13. CONTENT OF THE RESERVATION STATION FIFOS .... 27**~~
~~**14. THE OUTPUT FILE .... 27**~~
~~**15. IJ FORMAT .... 27**~~

~~15.1 Interpolation of constant attributes .... 28~~
~~16. STAGING REGISTERS .... 28~~
~~17. THE PARAMETER CACHE .... 30~~
~~18. VERTEX POSITION EXPORTING .... 31~~
~~19. EXPORTING ARBITRATION .... 31~~
~~20. EXPORTING RULES .... 31~~
~~20.1 Parameter caches exports .... 31~~
~~20.2 Memory exports .... 31~~
~~20.3 Position exports .... 31~~
~~21. EXPORT TYPES .... 31~~
~~21.1 Vertex Shading .... 31~~
~~21.2 Pixel Shading .... 31~~
~~22. SPECIAL INTERPOLATION MODES .... 31~~
~~22.1 Real time commands .... 31~~
~~22.2 Sprites/ XY screen coordinates/ FB information .... 32~~
~~22.3 Auto generated counters .... 32~~
~~22.3.1 Vertex shaders .... 32~~
~~22.3.2 Pixel shaders .... 32~~
~~23. STATE MANAGEMENT .... 33~~
~~23.1 Parameter cache synchronization .... 33~~
~~24. XY ADDRESS IMPORTS .... 33~~
~~24.1 Vertex indexes imports .... 33~~
~~25. REGISTERS .... 34~~
~~25.1 Control .... 34~~
~~25.2 Context .... 34~~
~~26. DEBUG REGISTERS .... 35~~
~~26.1 Context .... 35~~
~~26.2 Control .... 35~~
~~27. INTERFACES .... 35~~
~~27.1 External Interfaces .... 35~~
~~27.2 SC to SP Interfaces .... 35~~
~~27.2.1 SC_SP# .... 35~~
~~27.2.2 SC_SQ .... 36~~
~~27.2.3 SQ to SX: Interpolator bus .... 38~~
~~27.2.4 SQ to SP: Staging Register Data .... 38~~
~~27.2.5 VGT to SQ : Vertex interface .... 38~~
~~27.2.6 SQ to SX: Control bus .... 41~~
~~27.2.7 SX to SQ : Output file control .... 41~~
~~27.2.8 SQ to TP: Control bus .... 42~~
~~27.2.9 TP to SQ: Texture stall .... 42~~
~~27.2.10 SQ to SP: Texture stall .... 42~~
~~27.2.11 SQ to SP: GPR and auto counter .... 43~~
~~27.2.12 SQ to SPx: Instructions .... 44~~

~~27.2.13 SP to SQ: Constant address load/ Predicate Set.. 44~~
~~27.2.14 SQ to SPx: constant broadcast... 45~~
~~27.2.15 SP0 to SQ: Kill vector load... 45~~
~~27.2.16 SQ to CP: RBBM bus... 45~~
~~27.2.17 CP to SQ: RBBM bus... 45~~
~~27.2.18 SQ to CP: State report... 45~~
~~**28. OPEN ISSUES... 50**~~

Revision Changes:

Rev 0.1 (Laurent Lefebvre) Date: May 7, 2001	First draft.
Rev 0.2 (Laurent Lefebvre) Date : July 9, 2001	Changed the interfaces to reflect the changes in the SP. Added some details in the arbitration section.
Rev 0.3 (Laurent Lefebvre) Date : August 6, 2001	Reviewed the Sequencer spec after the meeting on August 3, 2001.
Rev 0.4 (Laurent Lefebvre) Date : August 24, 2001	Added the dynamic allocation method for register file and an example (written in part by Vic) of the flow of pixels/vertices in the sequencer.
Rev 0.5 (Laurent Lefebvre) Date : September 7, 2001	Added timing diagrams (Vic)
Rev 0.6 (Laurent Lefebvre) Date : September 24, 2001	Changed the spec to reflect the new R400 architecture. Added interfaces.
Rev 0.7 (Laurent Lefebvre) Date : October 5, 2001	Added constant store management, instruction store management, control flow management and data dependant predication.
Rev 0.8 (Laurent Lefebvre) Date : October 8, 2001	Changed the control flow method to be more flexible. Also updated the external interfaces.
Rev 0.9 (Laurent Lefebvre) Date : October 17, 2001	Incorporated changes made in the 10/18/01 control flow meeting. Added a NOP instruction, removed the conditional_execute_or_jump. Added debug registers.
Rev 1.0 (Laurent Lefebvre) Date : October 19, 2001	Refined interfaces to RB. Added state registers.
Rev 1.1 (Laurent Lefebvre) Date : October 26, 2001	Added SEQ→SP0 interfaces. Changed delta precision. Changed VGT→SP0 interface. Debug Methods added.
Rev 1.2 (Laurent Lefebvre) Date : November 16, 2001	Interfaces greatly refined. Cleaned up the spec.
Rev 1.3 (Laurent Lefebvre) Date : November 26, 2001	Added the different interpolation modes.
Rev 1.4 (Laurent Lefebvre) Date : December 6, 2001	Added the auto incrementing counters. Changed the VGT→SQ interface. Added content on constant management. Updated GPRs.
Rev 1.5 (Laurent Lefebvre) Date : December 11, 2001	Removed from the spec all interfaces that weren't directly tied to the SQ. Added explanations on constant management. Added PA→SQ synchronization fields and explanation.
Rev 1.6 (Laurent Lefebvre) Date : January 7, 2002	Added more details on the staging register. Added detail about the parameter caches. Changed the call instruction to a Conditionnal_call instruction. Added details on constant management and updated the diagram.
Rev 1.7 (Laurent Lefebvre) Date : February 4, 2002	Added Real Time parameter control in the SX interface. Updated the control flow section.
Rev 1.8 (Laurent Lefebvre) Date : March 4, 2002	New interfaces to the SX block. Added the end of clause modifier, removed the end of clause instructions.
Rev 1.9 (Laurent Lefebvre) Date : March 18, 2002	Rearangement of the CF instruction bits in order to ensure byte alignement.
Rev 1.10 (Laurent Lefebvre) Date : March 25, 2002	Updated the interfaces and added a section on exporting rules.
Rev 1.11 (Laurent Lefebvre) Date : April 19, 2002	Added CP state report interface. Last version of the spec with the old control flow scheme
Rev 2.0 (Laurent Lefebvre) Date : April 19, 2002	New control flow scheme

Rev 2.01 (Laurent Lefebvre) Date : May 2, 2002	Changed slightly the control flow instructions to allow force jumps and calls.

1. Overview

The sequencer chooses two ALU threads and a fetch hread to execute, and executes all of the instructions in a block before looking for a new clause of the same type. Two ALU threads are executed interleaved to hide the ALU latency. The arbitrator will give priority to older threads. There are two separate reservation stations, one for pixel vectors and one for vertices vectors. This way a pixel can pass a vertex and a vertex can pass a pixel.

To support the shader pipe the sequencer also contains the shader instruction cache, constant store, control flow constants and texture state. The four shader pipes also execute the same instruction thus there is only one sequencer for the whole chip.

The sequencer first arbitrates between vectors of 64 vertices that arrive directly from primitive assembly and vectors of 16 quads (64 pixels) that are generated in the scan converter.

The vertex or pixel program specifies how many GPRs it needs to execute. The sequencer will not start the next vector until the needed space is available in the GPRs.

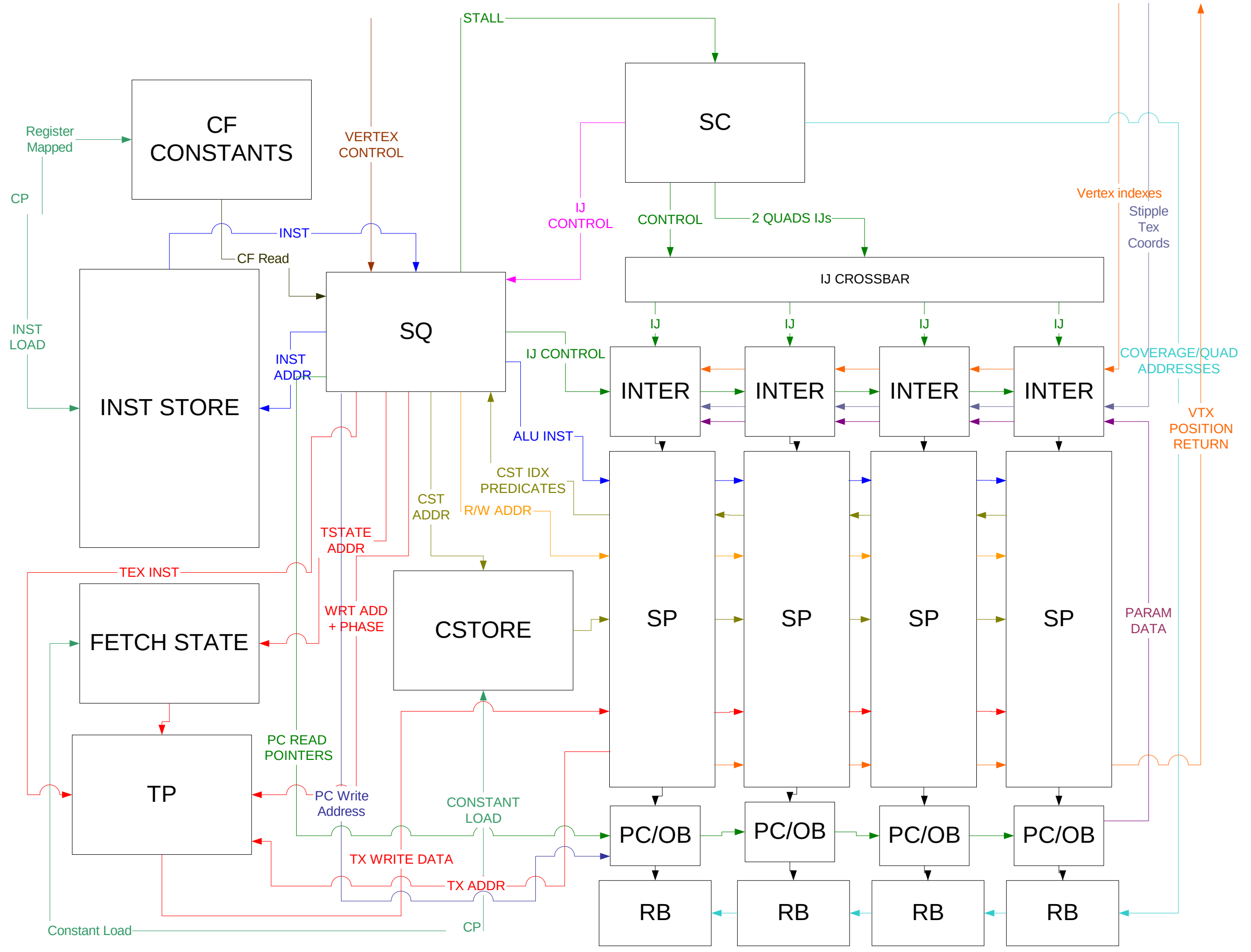


Figure 1: General Sequencer overview

1.1 Top Level Block Diagram

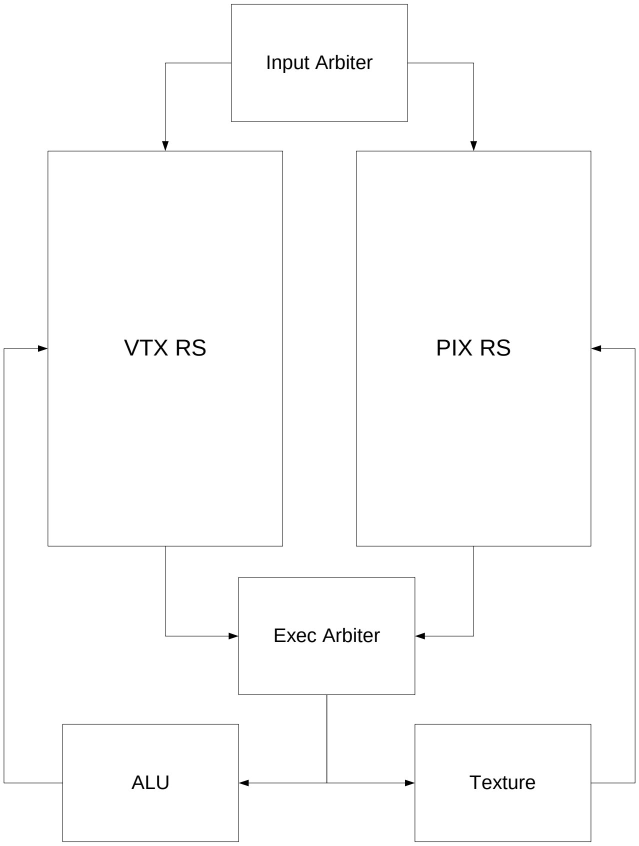


Figure 2: Reservation stations and arbiters

Under this new scheme, the sequencer (SQ) will only use one global state management machine per vector type (pixel, vertex) that we call the reservation station (RS).

1.2 Data Flow graph (SP)

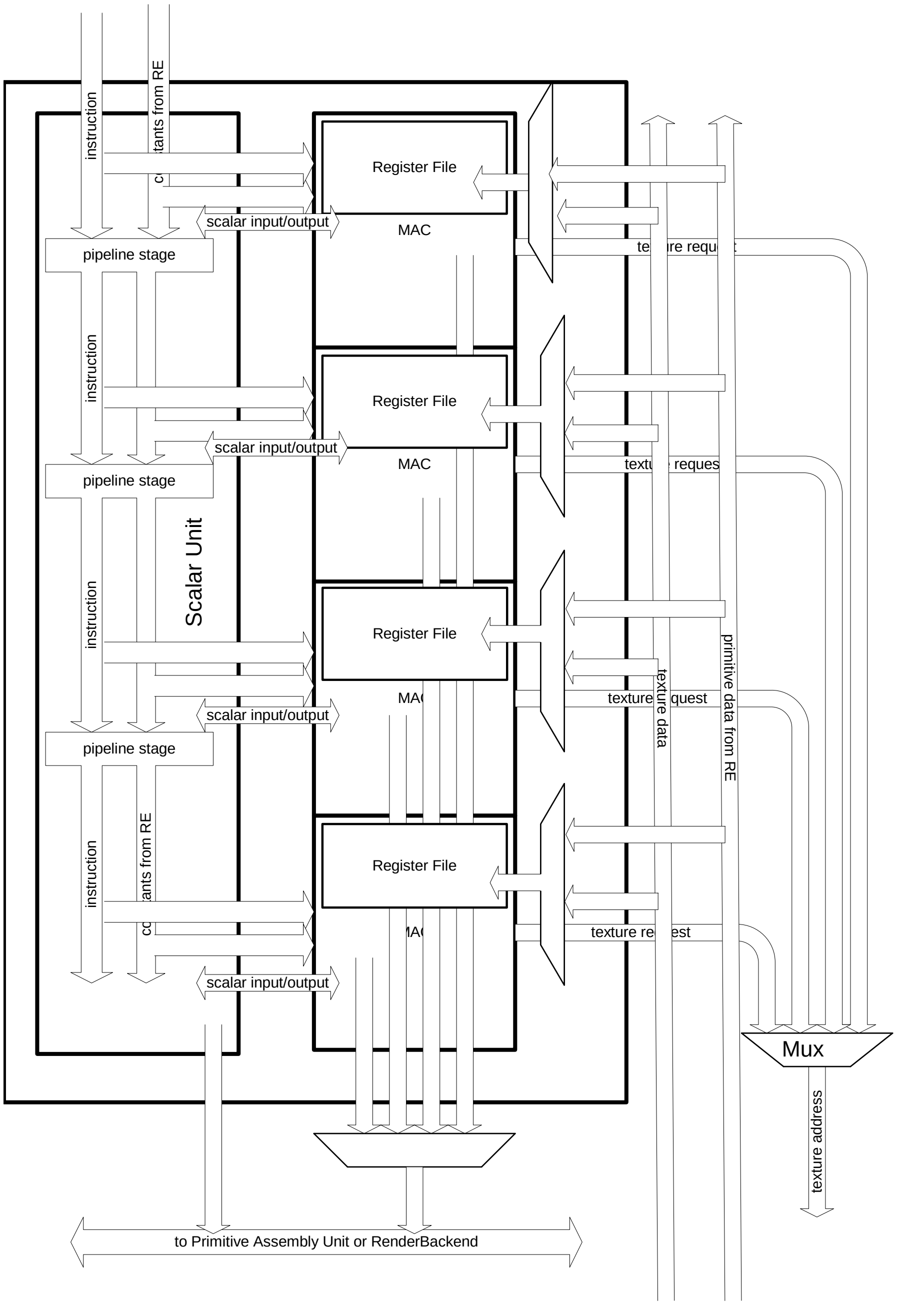


Figure 3: The shader Pipe

The gray area represents blocks that are replicated 4 times per shader pipe (16 times on the overall chip).

1.3 Control Graph

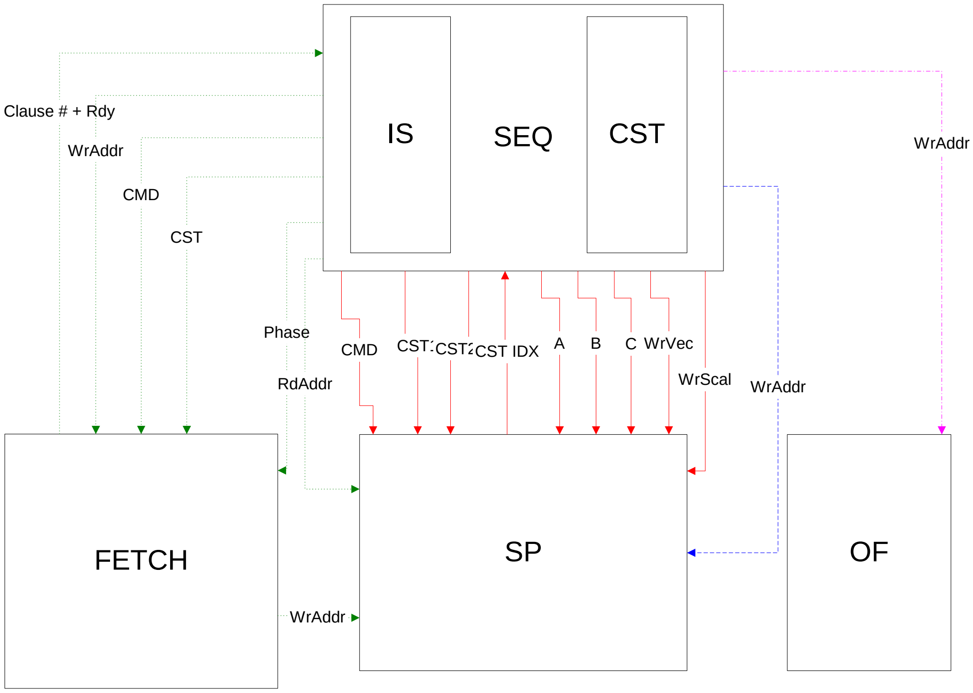


Figure 4: Sequencer Control interfaces

In green is represented the Fetch control interface, in red the ALU control interface, in blue the Interpolated/Vector control interface and in purple is the output file control interface.

2. Interpolated data bus

The interpolators contain an IJ buffer to pack the information as much as possible before writing it to the register file.

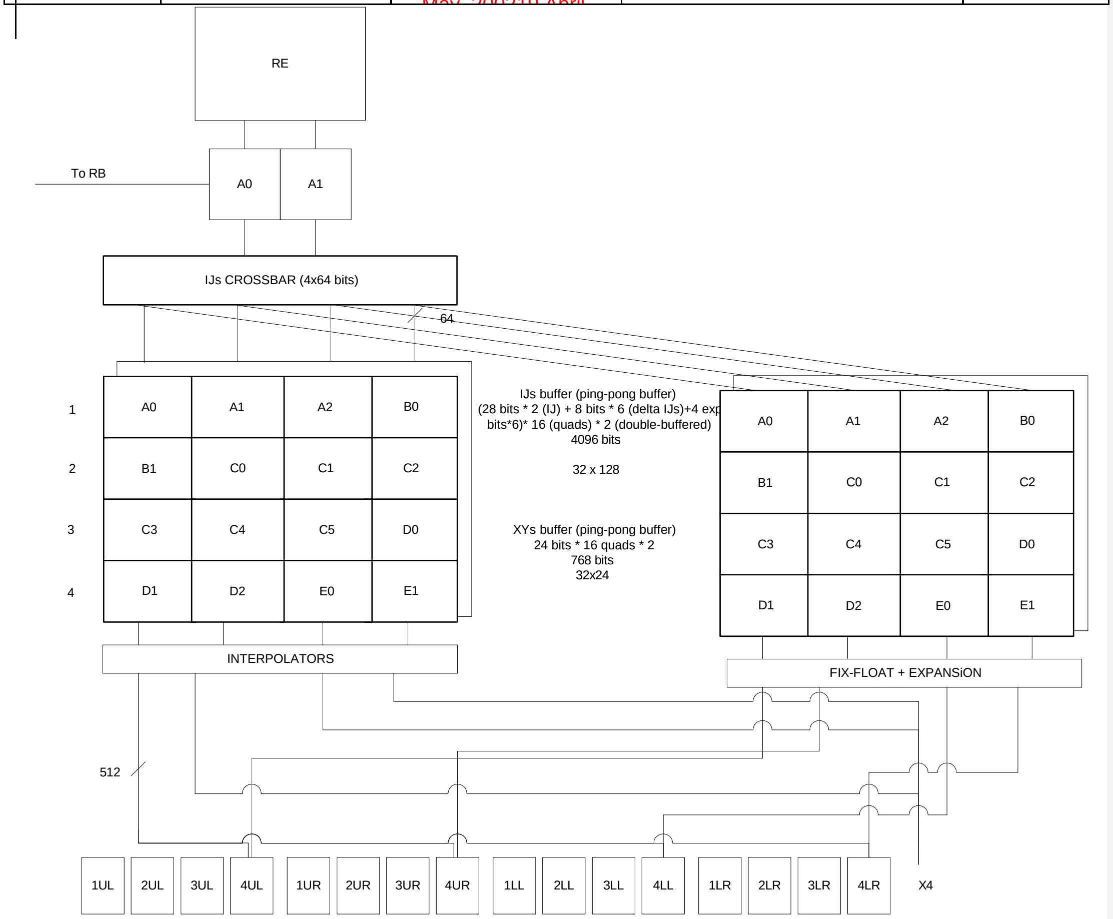

RE
To RB
A0
A1
IJs CROSSBAR (4x64 bits)
64
1
2
3
4
A0 A1 A2 B0
B1 C0 C1 C2
C3 C4 C5 D0
D1 D2 E0 E1
IJs buffer (ping-pong buffer)
(28 bits * 2 (IJ) + 8 bits * 6 (delta IJs)+4 exp
bits*6)* 16 (quads) * 2 (double-buffered)
4096 bits
32 x 128
XYs buffer (ping-pong buffer)
24 bits * 16 quads * 2
768 bits
32x24
INTERPOLATORS
FIX-FLOAT + EXPANSiON
512
1UL 2UL 3UL 4UL 1UR 2UR 3UR 4UR 1LL 2LL 3LL 4LL 1LR 2LR 3LR 4LR
X4

Figure 5: Interpolation buffers

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21	T22	T23
												WRITES												
SP 0	A0	A0	XY A0	B1	B1	XY B1	C3	C3	XY C3				D1	D1	XY D1									
SP 1	A1	A1	XY A1				C0	C0	XY C0	C4	C4	XY C4	D2	D2	XY D2									
SP 2	A2	A2	XY A2				C1	C1	XY C1	C5	C5	XY C5				E0	E0	XY E0						
SP 3				B0	B0	XY B0	C2	C2	XY C2				D0	D0	XY D0	E1	E1	XY E1						
												READS												
SP 0	XY 0-3	XY 16-19	XY 32-35	XY 48-51	A0	B1	C3	D1					A0	B1	C3	D1					V 0-3	V 16-19	V 32-35	V 48-51
SP 1	XY 4-7	XY 20-23	XY 36-39	XY 52-55	A1		C4	D2		C0			A1		C4	D2		C0			V 4-7	V 20-23	V 36-39	V 52-55
SP 2	XY 8-11	XY 24-27	XY 40-43	XY 56-59	A2		C5			C1		E0	A2		C5			C1		E0	V 8-11	V 24-27	V 40-43	V 56-59
SP 3	XY 12-15	XY 28-31	XY 44-47	XY 60-63					B0	C2	D0	E1					B0	C2	D0	E1	V 12-15	V 28-31	V 44-47	V 60-63
		XY						P1								P2						VTX		

Figure 6: Interpolation timing diagram

Above is an example of a tile the sequencer might receive from the SC. The write side is how the data get stacked into the XY and IJ buffers, the read side is how the data is passed to the GPRs. The IJ information is packed in the IJ buffer 4 quads at a time or two clocks. The sequencer allows at any given time as many as four quads to interpolate a parameter. They all have to come from the same primitive. Then the sequencer controls the write mask to the GPRs to write the valid data in.

3. Instruction Store

There is going to be only one instruction store for the whole chip. It will contain 4096 instructions of 96 bits each.

It is likely to be a 1 port memory; we use 1 clock to load the ALU instruction, 1 clocks to load the Fetch instruction, 1 clock to load 2 control flow instructions and 1 clock to write instructions.

The instruction store is loaded by the CP thru the register mapped registers.

The VS_BASE and PS_BASE context registers are used to specify for each context where its shader is in the instruction memory.

For the Real time commands the story is quite the same but for some small differences. There are no wrap-around points for real time so the driver must be careful not to overwrite regular shader data. The shared code (shared subroutines) uses the same path as real time.

4. Sequencer Instructions

All control flow instructions and move instructions are handled by the sequencer only. The ALUs will perform NOPs during this time (MOV PV,PV, PS,PS) if they have nothing else to do.

5. Constant Stores

5.1 Memory organizations

A likely size for the ALU constant store is 1024x128 bits. The read BW from the ALU constant store is 128 bits/clock and the write bandwidth is 32 bits/clock (directed by the CP bus size not by memory ports).

The maximum logical size of the constant store for a given shader is 256 constants. Or 512 for the pixel/vertex shader pair. The size of the re-mapping table is 128 lines (each line addresses 4 constants). The write granularity is 4 constants or 512 bits. It takes 16 clocks to write the four constants. Real time requires 256 lines in the physical memory (this is physically register mapped).

The texture state is also kept in a similar memory. The size of this memory is 320x96 bits (128 texture states for regular mode, 32 states for RT). The memory thus holds 128 texture states (192 bits per state). The logical size exposes 32 different states total, which are going to be shared between the pixel and the vertex shader. The size of the re-mapping table to for the texture state memory is 32 lines (each line addresses 1 texture state lines in the real memory). The CP write granularity is 1 texture state lines (or 192 bits). The driver sends 512 bits but the CP ignores the top 320 bits. It thus takes 6 clocks to write the texture state. Real time requires 32 lines in the physical memory (this is physically register mapped).

The control flow constant memory doesn't sit behind a renaming table. It is register mapped and thus the driver must reload its content each time there is a change in the control flow constants. Its size is 320*32 because it must hold 8 copies of the 32 dwords of control flow constants and the loop construct constants must be aligned.

The constant re-mapping tables for texture state and ALU constants are logically register mapped for regular mode and physically register mapped for RT operation.

5.2 Management of the Control Flow Constants

The control flow constants are register mapped, thus the CP writes to the according register to set the constant, the SQ decodes the address and writes to the block pointed by its current base pointer (CF_WR_BASE). On the read side, one level of indirection is used. A register (SQ_CONTEXT_MISC.CF_RD_BASE) keeps the current base pointer to the control flow block. This register is copied whenever there is a state change. Should the CP write to CF after the state change, the base register is updated with the (current pointer number +1)% number of states. This way, if the CP doesn't write to CF the state is going to use the previous CF constants.

5.3 Management of the re-mapping tables

5.3.1 *R400 Constant management*

The sequencer is responsible to manage two re-mapping tables (one for the constant store and one for the texture state). On a state change (by the driver), the sequencer will broadside copy the contents of its re-mapping tables to a new one. We have 8 different re-mapping tables we can use concurrently.

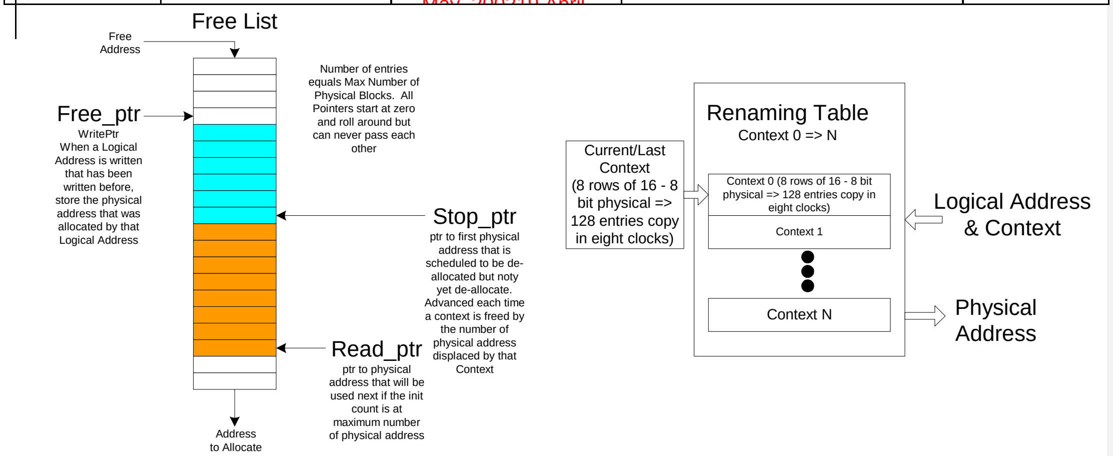
The constant memory update will be incremental, the driver only need to update the constants that actually changed between the two state changes.

For this model to work in its simplest form, the requirement is that the physical memory MUST be at least twice as large as the logical address space + the space allocated for Real Time. In our case, since the logical address space is 512 and the reserved RT space can be up to 256 entries, the memory must be of sizes 1280 and above. Similarly the size of the texture store must be of 32*2+32 = 96 entries and above.

5.3.2 *Proposal for R400LE constant management*

To make this scheme work with only 512+256 = 768 entries, upon reception of a CONTROL packet of state + 1, the sequencer would check for SQ_IDLE and PA_IDLE and if both are idle will erase the content of state to replace it with the new state (this is depicted in Figure 8: De-allocation mechanism~~Figure 8: De-allocation mechanismFigure 8: De-allocation mechanism~~). Note that in the case a state is cleared a value of 0 is written to the corresponding de-allocation counter location so that when the SQ is going to report a state change, nothing will be de-allocated upon the first report.

The second path sets all context dirty bits that were used in the current state to 1 (thus allowing the new state to reuse these physical addresses if needed).


Free List
Free Address
Free_ptr
WritePtr
When a Logical Address is written that has been written before, store the physical address that was allocated by that Logical Address
Number of entries equals Max Number of Physical Blocks. All Pointers start at zero and roll around but can never pass each other
Stop_ptr
ptr to first physical address that is scheduled to be de-allocated but noty yet de-allocate. Advanced each time a context is freed by the number of physical address displaced by that Context
Read_ptr
ptr to physical address that will be used next if the init count is at maximum number of physical address
Address to Allocate
Current/Last Context (8 rows of 16 - 8 bit physical => 128 entries copy in eight clocks)
Renaming Table
Context 0 => N
Context 0 (8 rows of 16 - 8 bit physical => 128 entries copy in eight clocks)
Context 1
Context N
Logical Address & Context
Physical Address

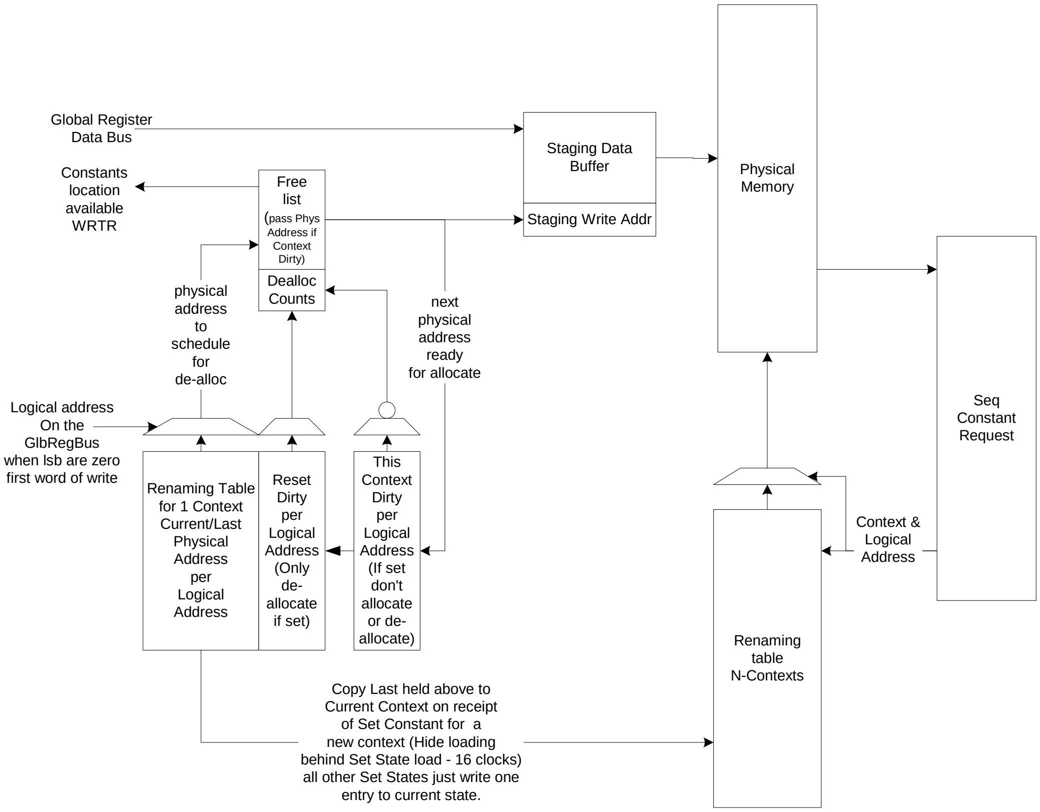

Global Register Data Bus
Staging Data Buffer
Staging Write Addr
Physical Memory
Constants location available WRTR
Free list (pass Phys Address if Context Dirty)
Dealloc Counts
physical address to schedule for de-alloc
next physical address ready for allocate
Seq Constant Request
Logical address On the GlbRegBus when lsb are zero first word of write
Renaming Table for 1 Context Current/Last Physical Address per Logical Address
Reset Dirty per Logical Address (Only de-allocate if set)
This Context Dirty per Logical Address (If set don't allocate or de-allocate)
Context & Logical Address
Renaming table N-Contexts
Copy Last held above to Current Context on receipt of Set Constant for a new context (Hide loading behind Set State load - 16 clocks) all other Set States just write one entry to current state.

Figure 7: Constant management

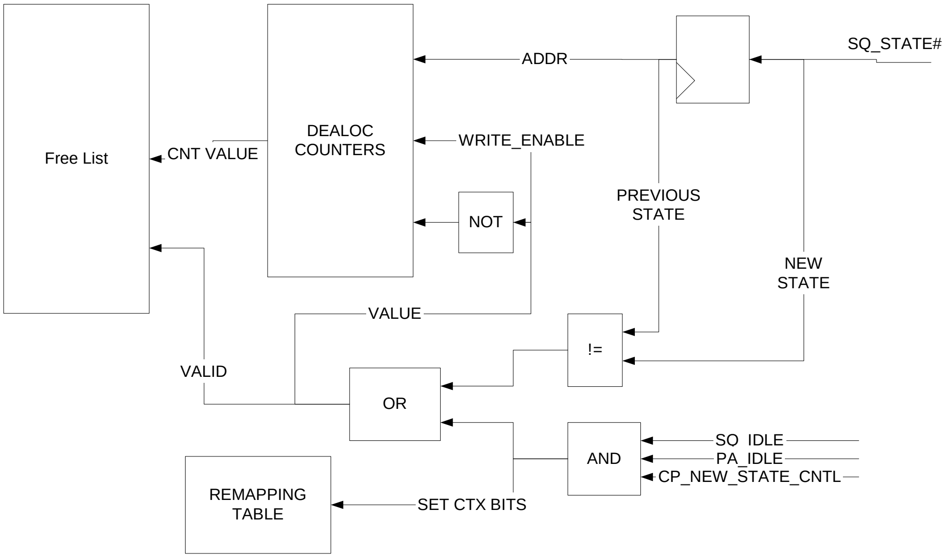


Figure 8: De-allocation mechanism for R400LE

5.3.3 *Dirty bits*

Two sets of dirty bits will be maintained per logical address. The first one will be set to zero on reset and set when the logical address is addressed. The second one will be set to zero whenever a new context is written and set for each address written while in this context. The reset dirty is not set, then writing to that logical address will not require de-allocation of whatever address stored in the renaming table. If it is set and the context dirty is not set, then the physical address store needs to be de-allocated and a new physical address is necessary to store the incoming data. If they are both set, then the data will be written into the physical address held in the renaming for the current logical address. No de-allocation or allocation takes place. This will happen when the driver does a set constant twice to the same logical address between context changes. NOTE: It is important to detect and prevent this, failure to do it will allow multiple writes to allocate all physical memory and thus hang because a context will not fit for rendering to start and thus free up space.

5.3.4 *Free List Block*

A free list block that would consist of a counter (called the IFC or Initial Free Counter) that would reset to zero and incremented every time a chunk of physical memory is used until they have all been used once. This counter would be checked each time a physical block is needed, and if the original ones have not been used up, us a new one, else check the free list for an available physical block address. The count is the physical address for when getting a chunk from the counter.

Storage of a free list big enough to store all physical block addresses.

Maintain three pointers for the free list that are reset to zero. The first one we will call write_ptr. This pointer will identify the next location to write the physical address of a block to be de-allocated. Note: we can never free more physical memory locations than we have. Once recording address the pointer will be incremented to walk the free list like a ring.

The second pointer will be called stop_ptr. The stop_ptr pointer will be advanced by the number of address chunks de-allocates when a context finishes. The address between the stop_ptr and write_ptr cannot be reused because they are still in use. But as soon as the context using then is dismissed the stop_ptr will be advanced.

The third pointer will be called read_ptr. This pointer will point will point to the next address that can be used for allocation as long as the read_ptr does not equal the stop_ptr and the IFC is at its maximum count.

5.3.5 *De-allocate Block*

This block will maintain a free physical address block count for each context. While in current context, a count shall be maintained specifying how many blocks were written into the free list at the write_ptr pointer. This count will be reset upon reset or when this context is active on the back and different than the previous context. It is actually a count of blocks in the previous context that will no longer be used. This count will be used to advance the write_ptr pointer to make available the set of physical blocks freed when the previous context was done. This allows the discard or de-allocation of any number of blocks in one clock.

5.3.6 *Operation of Incremental model*

The basic operation of the model would start with the write_ptr, stop_ptr, read_ptr pointers in the free list set to zero and the free list counter is set to zero. Also all the dirty bits and the previous context will be initialized to zero. When the first set constants happen, the reset dirty bit will not be set, so we will allocate a physical location from the free list counter because its not at the max value. The data will be written into physical address zero. Both the additional copy of the renaming table and the context zeros of the big renaming table will be updated for the logical address that was written by set start with physical address of 0. This process will be repeated for any logical address that are not dirty until the context changes. If a logical address is hit that has its dirty bits set while in the same context, both dirty bits would be set, so the new data will be over-written to the last physical address assigned for this logical address. When the first draw command of the context is detected, the previous context stored in the additional renaming table will be copied to the larger renaming table in the current (new) context location. Then the set constant logical address with be loaded with a new physical address during the copy and if the reset dirty was set, the physical address it replaced in the renaming table would be entered at the write_ptr pointer location on the free list and the write_ptr will be incremented. The de-allocation counter for the previous context (eight) will be incremented. This as set states come in for this context one of the following will happen:

1.) No dirty bits are set for the logical address being updated. A line will be allocated of the free-list counter or the free list at read_ptr pointer if read_ptr != to stop_ptr .
2.) Reset dirty set and Context dirty not set. A new physical address is allocated, the physical address in the renaming table is put on the free list at write_ptr and it is incremented along with the de-allocate counter for the last context.
3.) Context dirty is set then the data will be written into the physical address specified by the logical address.

This process will continue as long as set states arrive. This block will provide backpressure to the CP whenever he has not free list entries available (counter at max and stop_ptr == read_ptr). The command stream will keep a count of contexts of constants in use and prevent more than max constants contexts from being sent.

Whenever a draw packet arrives, the content of the re-mapping table is written to the correct re-mapping table for the context number. Also if the next context uses less constants than the current one all exceeding lines are moved to the free list to be de-allocated later. This happens in parallel with the writing of the re-mapping table to the correct memory.

Now preferable when the constant context leaves the last ALU clause it will be sent to this block and compared with the previous context that left. (Init to zero) If they differ than the older context will no longer be referenced and thus can be de-allocated in the physical memory. This is accomplished by adding the number of blocks freed this context to the stop_ptr pointer. This will make all the physical addresses used by this context available to the read_ptr allocate pointer for future allocation.

This device allows representation of multiple contexts of constants data with N copies of the logical address space. It also allows the second context to be represented as the first set plus some new additional data by just storing the delta's. It allows memory to be efficiently used and when the constants updates are small it can store multiple context. However, if the updates are large, less contexts will be stored and potentially performance will be degraded. Although it will still perform as well as a ring could in this case.

5.4 Constant Store Indexing

In order to do constant store indexing, the sequencer must be loaded first with the indexes (that come from the GPRs). There are 144 wires from the exit of the SP to the sequencer (9 bits pointers x 16 vertexes/clock). Since the data must pass thru the Shader pipe for the float to fixed conversion, there is a latency of 4 clocks (1 instruction)

between the time the sequencer is loaded and the time one can index into the constant store. The assembly will look like this

```
MOVA  R1.X,R2.X       // Loads the sequencer with the content of R2.X, also copies the content of R2.X into R1.X
NOP                   // latency of the float to fixed conversion
ADD   R3,R4,C0[R2.X]// Uses the state from the sequencer to add R4 to C0[R2.X] into R3
```

Note that we don't really care about what is in the brackets because we use the state from the MOVA instruction. R2.X is just written again for the sake of simplicity and coherency.

The storage needed in the sequencer in order to support this feature is 2*64*9 bits = 1152 bits.

5.5 Real Time Commands

The real time commands constants are written by the CP using the register mapped registers allocated for RT. It works is the same way than when dealing with regular constant loads BUT in this case the CP is not sending a logical address but rather a physical address and the reads are not passing thru the re-mapping table but are directly read from the memory. The boundary between the two zones is defined by the CONST_EO_RT control register. Similarly, for the fetch state, the boundary between the two zones is defined by the TSTATE_EO_RT control register.

5.6 Constant Waterfalling

In order to have a reasonable performance in the case of constant store indexing using the address register, we are going to have the possibility of using the physical memory port for read only. This way we can read 1 constant per clock and thus have a worst-case waterfall mode of 1 vertex per clock. There is a small synchronization issue related with this as we need for the SQ to make sure that the constants where actually written to memory (not only sent to the sequencer) before it can allow the first vector of pixels or vertices of the state to go thru the ALUs. To do so, the sequencer keeps 8 bits (one per render state) and sets the bits whenever the last render state is written to memory and clears the bit whenever a state is freed.

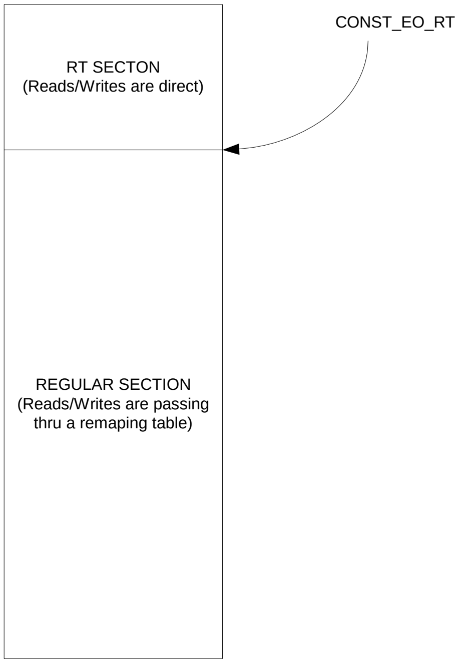


Figure 9: The instruction store

6. Looping and Branches

Loops and branches are planned to be supported and will have to be dealt with at the sequencer level. We plan on supporting constant loops and branches using a control program.

6.1 The controlling state.

The R400 controling state consists of:

Boolean[256:0]
Loop_count[7:0][31:0]
Loop_Start[7:0][31:0]
Loop_Step[7:0][31:0]

That is 256 Booleans and 32 loops.

We have a stack of 4 elements for nested calls of subroutines and 4 loop counters to allow for nested loops.

This state is available on a per shader program basis.

6.2 The Control Flow Program

We'd like to be able to code up a program of the form:

```
1:      Loop
2:      Exec    TexFetch
3:              TexFetch
4:              ALU
5:              ALU
6:              TexFetch
7:      End Loop
8:      ALU Export
```

But realize that 3: may be dependent on 2: and 4: is almost certainly dependent on 2: and 3:. Without clausing, these dependencies need to be expressed in the Control Flow instructions. Additionally, without separate 'texture clauses' and 'ALU clauses' we need to know which instructions to dispatch to the Texture Unit and which to the ALU unit. This information will be encapsulated in the flow control instructions.

Each control flow instruction will contain 2 bits of information for each (non-control flow) instruction:

a) ALU or Texture
b) Serialize Execution

(b) would force the thread to stop execution at this point (before the instruction is executed) and wait until all textures have been fetched. Given the allocation of reserved bits, this would mean that the count of an 'Exec' instruction would be limited to about 8 (non-control-flow) instructions. If more than this were needed, a second Exec (with the same conditions) would be issued.

Another function that relies upon 'clauses' is allocation and order of execution. We need to assure that pixels and vertices are exported in the correct order (even if not all execution is ordered) and that space in the output buffers are allocated in order. Additionally data can't be exported until space is allocated. A new control flow instruction:

Alloc <buffer select -- position,parameter, pixel or vertex memory. And the size required>.

would be created to mark where such allocation needs to be done. To assure allocation is done in order, the actual allocation for a given thread can not be performed unless the equivalent allocation for all previous threads is already completed. The implementation would also assure that execution of instruction(s) following the serialization due to the Alloc will occur in order -- at least until the next serialization or change from ALU to Texture. In most cases this will allow the exports to occur without any further synchronization. Only 'final' allocations or position allocations are

guaranteed to be ordered. Because strict ordering is required for pixels, parameters and positions, this implies only a single alloc for these structures. Vertex exports to memory do not require ordering during allocation and so multiple 'allocs' may be done.

6.2.1 *Control flow instructions table*

Here is the revised control flow instruction set.

Note that whenever a field is marked as RESERVED, it is assumed that all the bits of the field are cleared (0).

Execute					
47	46… 43	40 … 34	33 …16	15…12	11 … 0
Addressing	0001	RESERVED	Instructions type + serialize (9 instructions)	Count	Exec Address

Execute up to 9 instructions at the specified address in the instruction memory. The Instruction type field tells the sequencer the type of the instruction (LSB) (1 = Texture, 0 = ALU and whether to serialize or not the execution (MSB) (1 = Serialize, 0 = Non-Serialized).

NOP		
47	46 … 43	42 … 0
Addressing	0010	RESERVED

This is a regular NOP.

Conditional_Execute						
47	46 … 43	42	41 … 34	33…16	15 …12	11 … 0
Addressing	0011	Condition	Boolean address	Instructions type + serialize (9 instructions)	Count	Exec Address

If the specified Boolean (8 bits can address 256 Booleans) meets the specified condition then execute the specified instructions (up to 9 instructions). If the condition is not met, we go on to the next control flow instruction.

Conditional_Execute_Predicates							
47	46 … 43	42	41 … 36	35 … 34	33…16	15…12	11 … 0
Addressing	0010	Condition	RESERVED	Predicate vector	Instructions type + serialize (9 instructions)	Count	Exec Address

Check the AND/OR of all current predicate bits. If AND/OR matches the condition execute the specified number of instructions. We need to AND/OR this with the kill mask in order not to consider the pixels that aren't valid. If the condition is not met, we go on to the next control flow instruction.

Loop_Start					
47	46 … 43	42 … 17	20 … 16	15…12~~16 … 12~~	11 … 0
Addressing	0101	RESERVED	loop ID	RESERVED~~loop ID~~	Jump address

Loop Start. Compares the loop iterator with the end value. If loop condition not met jump to the address. Forward jump only. Also computes the index value. The loop id must match between the start to end, and also indicates which control flow constants should be used with the loop.

Loop_End						
47	46 … 43	42 … 2~~0~~4	23… 21	20 … 16~~19… 17~~	15…12~~16 … 12~~	11 … 0
Addressing	0011	RESERVED	Predicate break	loop ID ~~Predicate break~~	RESERVED ~~loop ID~~	start address

Loop end. Increments the counter by one, compares the loop count with the end value. If loop condition met, continue, else, jump BACK to the start of the loop. If predicate break != 0, then compares predicate vector n (specified by predicate break number). If all bits cleared then break the loop.

The way this is described does not prevent nested loops, and the inclusion of the loop id make this easy to do.

Conditionnal_Call						
47	46 … 43	42	~~35~~ 41 … 34	33 … 13~~2~~	12	11 … 0
Addressing	0111	Condition	~~Predicate vector~~Boolean address	RESERVED	Force Call	Jump address

If the condition is met, jumps to the specified address and pushes the control flow program counter on the stack. If force call is set the condition is ignored and the call is made always.

Return		
47	46 … 43	42 … 0
Addressing	1000	RESERVED

Pops the topmost address from the stack and jumps to that address. If nothing is on the stack, the program will just continue to the next instruction.

Conditionnal_Jump							
47	46 … 43	42	41… 34	33	32 … 13~~2~~	12	11 … 0
Addressing	1001	Condition	Boolean address	FW only	RESERVED	Force Jump	Jump address

If force jump is set the condition is ignored and the jump is made always. If FW only is set then only forward jumps are allowed.

Allocate				
47	46 … 43	42…41	40 … 4	3 …0
Debug	1010	Buffer Select	RESERVED	Allocation size

Buffer Select takes a value of the following:
01 – position export (ordered export)
10 – parameter cache or pixel export (ordered export)
11 – pass thru (out of order exports).

If debug is set this is a debug alloc (ignore if debug DB_ON register is set to off).

End Of Program		
47	46 … 43	42… 0
RESERVED	1011	RESERVED

Marks the end of the program.

6.3 Implementation

The envisioned implementation has a buffer that maintains the state of each thread. A thread lives in a given location in the buffer during its entire life, but the buffer has FIFO qualities in that threads leave in the order that they enter. Actually two buffers are maintained -- one for Vertices and one for Pixels. The intended implementation would allow for:

16 entries for vertices
48 entries for pixels.

From each buffer, arbitration logic attempts to select 1 thread for the texture unit and 1 (interleaved) thread for the ALU unit. Once a thread is selected it is read out of the buffer, marked as invalid, and submitted to appropriate execution unit. It is returned to the buffer (at the same place) with its status updated once all possible sequential instructions have been executed. A switch from ALU to TEX or visa-versa or a Serialize_Execution modifier forces the thread to be returned to the buffer.

Each entry in the buffer will be stored across two physical pieces of memory - most bits will be stored in a 1 read port device. Only bits needed for thread arbitration will be stored in a highly multi-ported structure. The bits kept in the 1 read port device will be termed 'state'. The bits kept in the multi-read ported device will be termed 'status'.

'State Bits' needed include:

1. Control Flow Instruction Pointer (~~12~~ 13 bits),
2. Execution Count Marker 4 bits),
3. Loop Iterators (4x9 bits),
4. Call return pointers (4x12 bits),
5. Predicate Bits (~~4x~~64 bits),
6. Export ID (1 bit),
7. Parameter Cache base Ptr (7 bits),
8. GPR Base Ptr (8 bits),
9. Context Ptr (3 bits).
10. LOD corrections (6x16 bits)
11. Valid bits (64 bits)

Formatted: Bullets and Numbering

Absent from this list are 'Index' pointers. These are costly enough that I'm presuming that they are instead stored in the GPRs. The first seven fields above (Control Flow Ptr, Execution Count, Loop Counts, call return ptrs, Predicate bits, PC base ptr and export ID) are updated every time the thread is returned to the buffer based on how much progress has been mode on thread execution. GPR Base Ptr, Context Ptr and LOD corrections are unchanged throughout execution of the thread.

'Status Bits' needed include:

- Valid Thread
- Texture/ALU engine needed
- Texture Reads are outstanding
- Waiting on Texture Read to Complete
- Allocation Wait (2 bits)
- 00 – No allocation needed
- 01 – Position export allocation needed (ordered export)
- 10 – Parameter or pixel export needed (ordered export)
- 11 – pass thru (out of order export)
- Allocation Size (4 bits)
- Position Allocated
- First thread of a new context
- Event thread (NULL thread that needs to trickle down the pipe)
- Last (1 bit)
- Pulse SX (1 bit)

Formatted: Bullets and Numbering

All of the above fields from all of the entries go into the arbitration circuitry. The arbitration circuitry will select a winner for both the Texture Engine and for the ALU engine. There are actually two sets of arbitration -- one for pixels and one for vertices. A final selection is then done between the two. But the rest of this implementation summary only considers the 'first' level selection which is similar for both pixels and vertices.

Texture arbitration requires no allocation or ordering so it is purely based on selecting the 'oldest' thread that requires the Texture Engine.

ALU arbitration is a little more complicated. First, only threads where either of Texture_Reads_outstanding or Waiting_on_Texture_Read_to_Complete are '0' are considered. Then if Allocation_Wait is active, these threads are further filtered based on whether space is available. If the allocation is position allocation, then the thread is only considered if all 'older' threads have already done their position allocation (position allocated bits set). If the allocation is parameter or pixel allocation, then the thread is only considered if it is the oldest thread. Also a thread is not considered if it is a parameter or pixel or position allocation, has its First_thread_of_a_new_context bit set and would cause ALU interleaving with another thread performing the same parameter or pixel or position allocation. Finally the 'oldest' of the threads that pass through the above filters is selected. If the thread needed to allocate, then at this time the allocation is done, based on Allocation_Size. If a thread has its “last” bit set, then it is also removed from the buffer, never to return.

If I now redefine 'clauses' to mean 'how many times the thread is removed from the thread buffer for the purpose of exection by either the ALU or Texture engine', then the minimum number of clauses needed is 2 -- one to perform the allocation for exports (execution automatically halts after an 'Alloc' instruction) (but doesn't performs the actual allocation) and one for the actual ALU/export instructions. As the 'Alloc' instruction could be part of a texture clause (presumably the final instruction in such a clause), a thread could still execute in this minimal number of 2 clauses, even if it involved texture fetching.

The Texture_Reads_Outstanding bit must be updated by the sequencer, based on keeping track of how many Texture Clauses have been executed by a given thread that have not yet had there data returned. Any number above 0 results in this bit being set. We could consider forcing synchronization such that two texture clauses for a given thread may not be outstanding at any time (that would be my preference for simplicity reasons and because it would require only very little change in the texture pipe interface). This would allow the sequencer to set the bit on execution of the texture clause, and allow the texture unit to return a pointer to the thread buffer on completion that clears the bit.

6.4 Data dependant predicate instructions

Data dependant conditionals will be supported in the R400. The only way we plan to support those is by supporting three vector/scalar predicate operations of the form:

PRED_SETE_# - similar to SETE except that the result is 'exported' to the sequencer.
PRED_SETNE_# - similar to SETNE except that the result is 'exported' to the sequencer.
PRED_SETGT_# - similar to SETGT except that the result is 'exported' to the sequencer
PRED_SETGTE_# - similar to SETGTE except that the result is 'exported' to the sequencer

For the scalar operations only we will also support the two following instructions:
PRED_SETE0_# – SETE0
PRED_SETE1_# – SETE1

The export is a single bit - 1 or 0 that is sent using the same data path as the MOVA instruction. The sequencer will maintain 4 sets of 64 bit predicate vectors (in fact 8 sets because we interleave two programs but only 4 will be exposed) and use it to control the write masking. This predicate is not maintained across clause boundaries. The # sign is used to specify which predicate set you want to use 0 thru 3.

Then we have two conditional execute bits. The first bit is a conditional execute “on” bit and the second bit tells us if we execute on 1 or 0. For example, the instruction:

P0_ADD_# R0,R1,R2

Is only going to write the result of the ADD into those GPRs whose predicate bit is 0. Alternatively, P1_ADD_# would only write the results to the GPRs whose predicate bit is set. The use of the P0 or P1 without precharging the sequencer with a PRED instruction is undefined.

{Issue: do we have to have a NOP between PRED and the first instruction that uses a predicate?}

6.5 HW Detection of PV,PS

Because of the control program, the compiler cannot detect statically dependant instructions. In the case of non-masked writes and subsequent reads the sequencer will insert uses of PV,PS as needed. This will be done by comparing the read address and the write address of consecutive instructions. For masked writes, the sequencer will insert NOPs wherever there is a dependant read/write.

The sequencer will also have to insert NOPs between PRED_SET and MOVA instructions and their uses.

6.6 Register file indexing

Because we can have loops in fetch clause, we need to be able to index into the register file in order to retrieve the data created in a fetch clause loop and use it into an ALU clause. The instruction will include the base address for register indexing and the instruction will contain these controls:

Bit7	Bit 6	
0	0	'absolute register'
0	1	'relative register'
1	0	'previous vector'
1	1	'previous scalar'

In the case of an absolute register we just take the address as is. In the case of a relative register read we take the base address and we add to it the loop_index and this becomes our new address that we give to the shader pipe.

The sequencer is going to keep a loop index computed as such:

Index = Loop_iterator*Loop_step + Loop_start.

We loop until loop_iterator = loop_count. Loop_step is a signed value [-128…127]. The computed index value is a 10 bit counter that is also signed. Its real range is [-256,256]. The tenth bit is only there so that we can provide an out of range value to the "indexing logic" so that it knows when the provided index is out of range and thus can make the necessary arrangements.

6.7 Debugging the Shaders

In order to be able to debug the pixel/vertex shaders efficiently, we provide 2 methods.

6.7.1 *Method 1: Debugging registers*

Current plans are to expose 2 debugging, or error notification, registers:
1. address register where the first error occurred
2. count of the number of errors

The sequencer will detect the following groups of errors:
- count overflow
- constant indexing overflow
- register indexing overflow

Compiler recognizable errors:
- jump errors
    - relative jump address > size of the control flow program
- call stack
    - call with stack full
    - return with stack empty

A jump error will always cause the program to break. In this case, a break means that a clause will halt execution, but allowing further clauses to be executed.

With all the other errors, program can continue to run, potentially to worst-case limits. The program will only break if the DB_PROB_BREAK register is set.

If indexing outside of the constant or the register range, causing an overflow error, the hardware is specified to return the value with an index of 0. This could be exploited to generate error tokens, by reserving and initializing the 0th register (or constant) for errors.

{ISSUE : Interrupt to the driver or not?}

6.7.2 *Method 2: Exporting the values in the GPRs*

1) The sequencer will have a debug active, count register and an address register for this mode.

Under the normal mode execution follows the normal course.

Under the debug mode it is assumed that the program is always exporting n debug vectors and that all other exports to the SX block (position, color, z, ect) will been turned off (changed into NOPs) by the sequencer (even if they occur before the address stated by the ADDR debug register).

7. Pixel Kill Mask

A vector of 64 bits is kept by the sequencer per group of pixels/vertices. Its purpose is to optimize the texture fetch requests and allow the shader pipe to kill pixels using the following instructions:

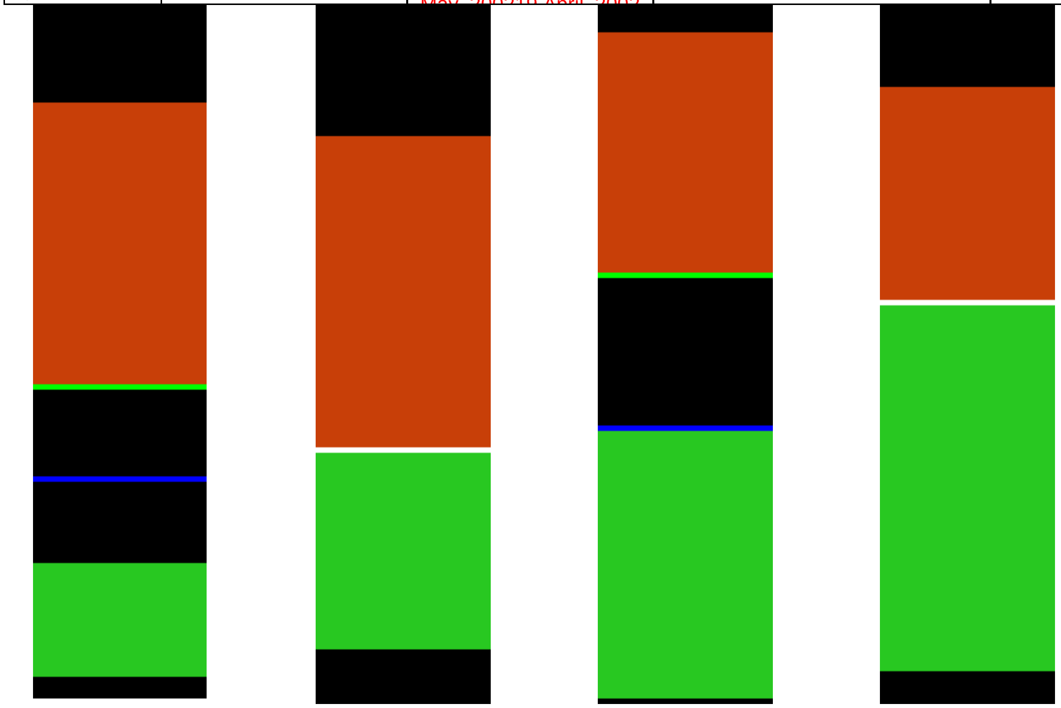
MASK_SETE
MASK_SETNE
MASK_SETGT
MASK_SETGTE

8. Multipass vertex shaders (HOS)

Multipass vertex shaders are able to export from the 6 last clauses but to memory ONLY.

9. Register file allocation

The register file allocation for vertices and pixels can either be static or dynamic. In both cases, the register file in managed using two round robins (one for pixels and one for vertices). In the dynamic case the boundary between pixels and vertices is allowed to move, in the static case it is fixed to 128-VERTEX_REG_SIZE for vertices and PIXEL_REG_SIZE for pixels.



Above is an example of how the algorithm works. Vertices come in from top to bottom; pixels come in from bottom to top. Vertices are in orange and pixels in green. The blue line is the tail of the vertices and the green line is the tail of the pixels. Thus anything between the two lines is shared. When pixels meets vertices the line turns white and the boundary is static until both vertices and pixels share the same “unallocated bubble”. Then the boundary is allowed to move again. The numbering of the GPRs starts from the bottom of the picture at index 0 and goes up to the top at index 127.

10. Fetch Arbitration

The fetch arbitration logic chooses one of the 8 potentially pending fetch clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute. Once chosen, the clause state machine will send one 2x2 fetch per clock (or 4 fetches in one clock every 4 clocks) until all the fetch instructions of the clause are sent. This means that there cannot be any dependencies between two fetches of the same clause.

The arbitrator will not wait for the fetches to return prior to selecting another clause for execution. The fetch pipe will be able to handle up to X(?) in flight fetches and thus there can be a fair number of active clauses waiting for their fetch return data.

11. ALU Arbitration

ALU arbitration proceeds in almost the same way than fetch arbitration. The ALU arbitration logic chooses one of the 8 potentially pending ALU clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute. There are two ALU arbiters, one for the even clocks and one for the odd clocks. For example, here is the sequencing of two interleaved ALU clauses (E and O stands for Even and Odd sets of 4 clocks):

Einst0 Oinst0 Einst1 Oinst1 Einst2 Oinst2 Einst0 Oinst3 Einst1 Oinst4 Einst2 Oinst0…
Proceeding this way hides the latency of 8 clocks of the ALUs. Also note that the interleaving also occurs across clause boundaries.

12. Handling Stalls

When the output file is full, the sequencer prevents the ALU arbitration logic from selecting the last clause (this way nothing can exit the shader pipe until there is place in the output file. If the packet is a vertex packet and the position buffer is full (POS_FULL) then the sequencer also prevents a thread from entering the exporting clause (3?). The sequencer will set the OUT_FILE_FULL signal n clocks before the output file is actually full and thus the ALU arbiter will be able read this signal and act accordingly by not preventing exporting clauses to proceed.

13. Content of the reservation station FIFOs

The reservation FIFOs contain the state of the vector of pixels and vertices. We have two sets of those: one for pixels, and one for vertices. They contain 3 bits of Render State 7 bits for the base address of the GPRs, some bits for LOD correction and coverage mask information in order to fetch fetch for only valid pixels, the quad address.

14. The Output File

The output file is where pixels are put before they go to the RBs. The write BW to this store is 256 bits/clock. Just before this output file are staging registers with write BW 512 bits/clock and read BW 256 bits/clock. The staging registers are 4x128 (and there are 16 of those on the whole chip).

15. IJ Format

The IJ information sent by the PA is of this format on a per quad basis:

We have a vector of IJ's (one IJ per pixel at the centroid of the fragment or at the center of the pixel depending on the mode bit). The interpolation is done at a different precision across the 2x2. The upper left pixel's parameters are always interpolated at full 20x24 mantissa precision. Then the result of the interpolation along with the difference in IJ in reduced precision is used to interpolate the parameter for the other three pixels of the 2x2. Here is how we do it:

Assuming P0 is the interpolated parameter at Pixel 0 having the barycentric coordinates I(0), J(0) and so on for P1,P2 and P3. Also assuming that A is the parameter value at V0 (interpolated with I), B is the parameter value at V1 (interpolated with J) and C is the parameter value at V2 (interpolated with (1-I-J).

$$\Delta 01I = I(1) - I(0)$$
$$\Delta 01J = J(1) - J(0)$$
$$\Delta 02I = I(2) - I(0)$$
$$\Delta 02J = J(2) - J(0)$$
$$\Delta 03I = I(3) - I(0)$$
$$\Delta 03J = J(3) - J(0)$$

P0	P1
P2	P3

$$P0 = C + I(0) * (A - C) + J(0) * (B - C)$$
$$P1 = P0 + \Delta 01I * (A - C) + \Delta 01J * (B - C)$$
$$P2 = P0 + \Delta 02I * (A - C) + \Delta 02J * (B - C)$$
$$P3 = P0 + \Delta 03I * (A - C) + \Delta 03J * (B - C)$$

P0 is computed at 20x24 mantissa precision and P1 to P3 are computed at 8X24 mantissa precision. So far no visual degradation of the image was seen using this scheme.

Multiplies (Full Precision): 2
Multiplies (Reduced precision): 6
Subtracts 19x24 (Parameters): 2

Adds: 8

FORMAT OF P0's IJ : Mantissa 20 Exp 4 for I + Sign
Mantissa 20 Exp 4 for J + Sign

FORMAT of Deltas (x3):Mantissa 8 Exp 4 for I + Sign
Mantissa 8 Exp 4 for J + Sign

Total number of bits : 20*2 + 8*6 + 4*8 + 4*2 = 128

All numbers are kept using the un-normalized floating point convention: if exponent is different than 0 the number is normalized if not, then the number is un-normalized. The maximum range for the IJs (Full precision) is +/- 63 and the range for the Deltas is +/- 127.

15.1 Interpolation of constant attributes

Because of the floating point imprecision, we need to take special provisions if all the interpolated terms are the same or if two of the barycentric coordinates are the same.

We start with the premise that if A = B and B = C and C = A, then P0,1,2,3 = A. Since one or more of the IJ terms may be zero, so we extend this to:

```
if (A=B and B=C and C=A)
   P0,1,2,3 = A;
else if ((I = 0) or (J = 0)) and
          ((J = 0) or (1-I-J = 0)) and
          ((1-J-I = 0) or (I = 0))) {
                if(I != 0) {
                    P0 = A;
                } else if(J != 0) {
                    P0 = B;
                } else {
                    P0 = C;
               }
             //rest of the quad interpolated normally
}
else
{
             normal interpolation
}
```

16. Staging Registers

In order for the reuse of the vertices to be 14, the sequencer will have to re-order the data sent IN ORDER by the VGT for it to be aligned with the parameter cache memory arrangement. Given the following group of vertices sent by the VGT:

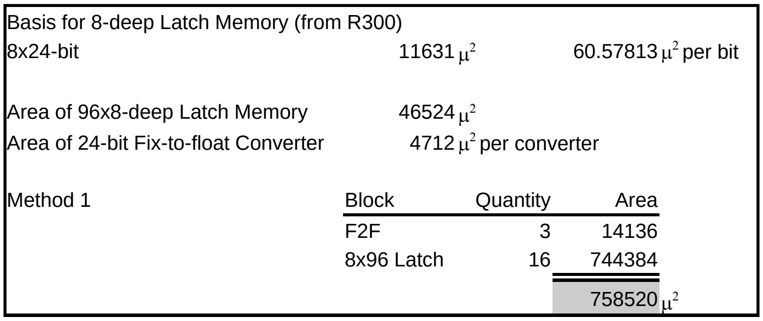
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 || 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 || 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 || 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63

The sequencer will re-arrange them in this fashion:

0 1 2 3 16 17 18 19 32 33 34 35 48 49 50 51 || 4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55 || 8 9 10 11 24 25 26 27 40 41 42 43 56 57 58 59 || 12 13 14 15 28 29 30 31 44 45 46 47 60 61 62 63

The || markers show the SP divisions. In the event a shader pipe is broken, the VGT will send padding to account for the missing pipe. For example, if SP1 is broken, vertices 4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55 will still be sent by the VGT to the SQ **BUT** will not be processed by the SP and thus should be considered invalid (by the SU and VGT).

The most straightforward, *non-compressed* interface method would be to convert, in the VGT, the data to 32-bit floating point prior to transmission to the VSISRs. In this scenario, the data would be transmitted to (and stored in) the VSISRs in full 32-bit floating point. This method requires three 24-bit fixed-to-float converters in the VGT. Unfortunately, it also requires and additional 3,072 bits of storage across the VSISRs. This interface is illustrated in Figure 11~~Figure 11~~~~Figure 11~~. The area of the fixed-to-float converters and the VSISRs for this method is roughly estimated as 0.759sqmm using the R300 process. The gate count estimate is shown in Figure 10~~Figure 10~~~~Figure 10~~.



Basis for 8-deep Latch Memory (from R300)			
8x24-bit	11631 μ²		60.57813 μ² per bit
Area of 96x8-deep Latch Memory	46524 μ²		
Area of 24-bit Fix-to-float Converter	4712 μ² per converter		

Method 1	Block	Quantity	Area
	F2F	3	14136
	8x96 Latch	16	744384
			758520 μ²

Figure 10:Area Estimate for VGT to Shader Interface

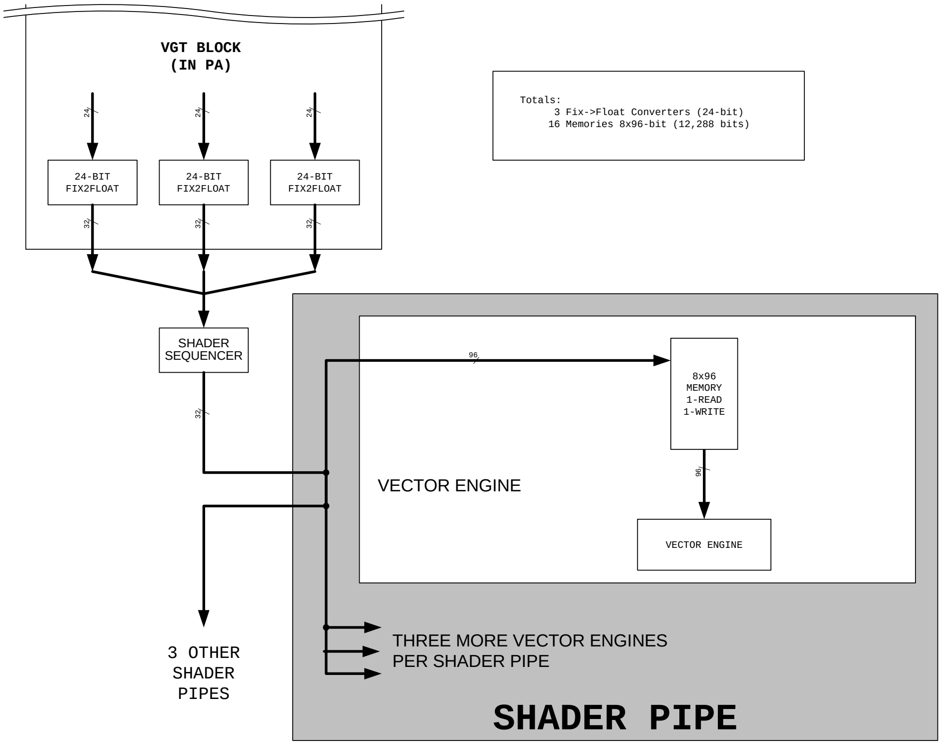


Figure 11:VGT to Shader Interface

17. The parameter cache

The parameter cache is where the vertex shaders export their data. It consists of 16 128x128 memories (1R/1W). The reuse engine will make it so that all vertexes of a given primitive will hit different memories. The allocation method for these memories is a simple round robin. The parameter cache pointers are mapped in the following way: 4MSBs are the memory number and the 7 LSBs are the address within this memory.

MEMORY NUMBER 4 bits	ADDRESS 7 bits

The PA generates the parameter cache addresses as the positions come from the SQ. All it needs to do is keep a Current_Location pointer (7 bits only) and as the positions comes increment the memory number. When the memory number field wraps around, the PA increments the Current_Location by VS_EXPORT_COUNT (a snooped register from the SQ). As an example, say the memories are all empty to begin with and the vertex shader is exporting 8 parameters per vertex (VS_EXPORT_COUNT = 8). The first position received is going to have the PC address 00000000000 the second one 00010000000, third one 00100000000 and so on up to 11110000000. Then the next position received (the 17th) is going to have the address 00000001000, the 18th 00010001000, the 19th 00100001000 and so on. The Current_location is NEVER reset BUT on chip resets. The only thing to be careful about is that if the SX doesn't send you a full group of positions (<64) then you need to fill the address space so that the next group starts correctly aligned (for example if you receive only 33 positions then you need to add 2*VS_EXPORT_COUNT to Current_Location and reset the memory count to 0 before the next vector begins).

17.1 Export restrictions

17.1.1 *Pixel exports:*

Pixels can export 1,2,3 or 4 color buffers to the SX(+z). The exports will be done in order. The PRED_OPTIMIZE function has to be turned of if the exports are done using interleaved predicated instructions. The exports will always be ordered to the SX.

17.1.2 *Vertex exports:*

Position or parameter caches can be exported in any order in the shader program. It is always better to export posistion as soon as possible. Position has to be exported in a single export block (no texture instructions can be placed between the exports). Parameter cache exports can be done in any order with texture instructions interleaved. The PRED_OPTIMIZE function has to be turned of if the exports are done using interleaved predicated instructions to the Parameter cache (see Arbitration restrictions for details). The exports will always be allocated in order to the SX.

17.1.3 *Pass thru exports:*

Pass thru exports have to be done in groups of the form:

```
Alloc 4 (8 or 12)
Execute ALU(ADDR) ALU(DATA) ALU(DATA) ALU(DATA)…
```

They cannot have texture instructions interleaved in the export block. These exports **are not guaranteed to be ordered**.

Also, when doing a pass thru export, Position MUST be exported AFTER all pass thru exports. This position export is used to synchronize the chip when doing a transition from pass thru shader to regular shader and vice versa.

17.2 Arbitration restrictions

Here are the Sequencer arbitration restrictions:

1) Cannot execute a serialized thread if the corresponding texture pending bit is set
2) Cannot allocate position if any older thread has not allocated position
3) If last thread is marked as not valid AND marked as last and we are about to execute the second to oldest thread also marked last then:
    a. Both threads must be from the same context (cannot allow a first thread)
    b. Must turn off the predicate optimization for the second thread
4) Cannot execute a texture clause if texture reads are pending
5) Cannot execute last if texture pending (even if not serial)

18. Export Types

The export type (or the location where the data should be put) is specified using the destination address field in the ALU instruction. Here is a list of all possible export modes:

18.1 Vertex Shading

0:15	- 16 parameter cache
16:31	- Empty (Reserved?)
32	- Export Address
33:40	- 8 vertex exports to the frame buffer and index
41:47	- Empty
48:55	- 8 debug export (interpret as normal vertex export)
60	- export addressing mode
61	- Empty
62	- position

63	- sprite size export that goes with position export (point_h,point_w,edgeflag,misc)

18.2 Pixel Shading

0	- Color for buffer 0 (primary)
1	- Color for buffer 1
2	- Color for buffer 2
3	- Color for buffer 3
4:7	- Empty
8	- Buffer 0 Color/Fog (primary)
9	- Buffer 1 Color/Fog
10	- Buffer 2 Color/Fog
11	- Buffer 3 Color/Fog
12:15	- Empty
16:31	- Empty (Reserved?)
32	- Export Address
33:40	- 8 exports for multipass pixel shaders.
41:47	- Empty
48:55	- 8 debug exports (interpret as normal pixel export)
60	- export addressing mode
61:62	- Empty
63	- Z for primary buffer (Z exported to 'alpha' component)

19. Special Interpolation modes

19.1 Real time commands

We are unable to use the parameter memory since there is no way for a command stream to write into it. Instead we need to add three 16x128 memories (one for each of three vertices x 16 interpolants). These will be mapped onto the register bus and written by type 0 packets, and output to the the parameter busses (the sequencer and/or PA need to be able to address the realtime parameter memory as well as the regular parameter store. For higher performance we should be able able to view them as two banks of 16 and do double buffering allowing one to be loaded, while the other is rasterized with. Most overlay shaders will need 2 or 4 scalar coordinates, one option might be to restrict the memory to 16x64 or 32x64 allowing only two interpolated scalars per cycle, the only problem I see with this is, if we view support for 16 vector-4 interpolants important (true only if we map Microsoft's high priority stream to the realtime stream), then the PA/sequencer need to support a realtime-specific mode where we need to address 32 vectors of parameters instead of 16. This mode is triggered by the primitive type: REAL TIME. The actual memories are in the in the SX blocks. The parameter data memories are hooked on the RBBM bus and are loaded by the CP using register mapped memory.

19.2 Sprites/ XY screen coordinates/ FB information

When working with sprites, one may want to overwrite the parameter 0 with SC generated data. Also, XY screen coordinates may be needed in the shader program. This functionality is controlled by the gen_I0 register (in SQ) in conjunction with the SND_XY register (in SC). Also it is possible to send the faceness information (for OGL front/back special operations) to the shader using the same control register. Here is a list of all the modes and how they interact together:

Gen_st is a bit taken from the interface between the SC and the SQ. This is the MSB of the primitive type. If the bit is set, it means we are dealing with Point AA, Line AA or sprite and in this case the vertex values are going to generated between 0 and 1.

Param_Gen_I0 disable, snd_xy disable, no gen_st – I0 = No modification
Param_Gen_I0 disable, snd_xy disable, gen_st – I0 = No modification
Param_Gen_I0 disable, snd_xy enable, no gen_st – I0 = No modification
Param_Gen_I0 disable, snd_xy enable, gen_st – I0 = No modification
Param_Gen_I0 enable, snd_xy disable, no gen_st – I0 = garbage, garbage, garbage, faceness

Param_Gen_I0 enable, snd_xy disable, gen_st – I0 = garbage, garbage, s, t
Param_Gen_I0 enable, snd_xy enable, no gen_st – I0 = screen x, screen y, garbage, faceness
Param_Gen_I0 enable, snd_xy enable, gen_st – I0 = screen x, screen y, s, t

19.3 Auto generated counters

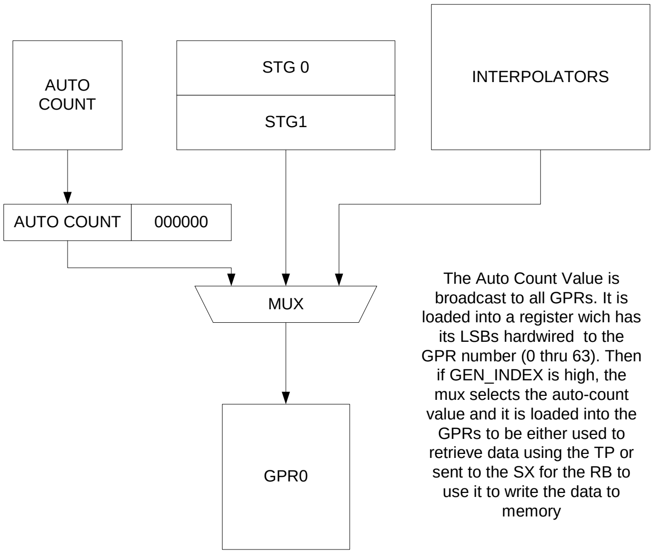
In the cases we are dealing with multipass shaders, the sequencer is going to generate a vector count to be able to both use this count to write the 1st pass data to memory and then use the count to retrieve the data on the 2nd pass. The count is always generated in the same way but it is passed to the shader in a slightly different way depending on the shader type (pixel or vertex). This is toggled on and off using the GEN_INDEX register. The sequencer is going to keep two counters, one for pixels and one for vertices. Every time a full vector of vertices or pixels is written to the GPRs the counter is incremented. Every time a state change is detected, the corresponding counter is reset. While there is only one count broadcast to the GPRs, the LSB are hardwired to specific values making the index different for all elements in the vector.

19.3.1 *Vertex shaders*

In the case of vertex shaders, if GEN_INDEX is set, the data will be put into the x field of the third register (it means that the compiler must allocate 3 GPRs in all multipass vertex shader modes).

19.3.2 *Pixel shaders*

In the case of pixel shaders, if GEN_INDEX is set and Param_Gen_I0 is enabled, the data will be put in the x field of the 2nd register (R1.x), else if GEN_INDEX is set the data will be put into the x field of the 1st register (R0.x).



The Auto Count Value is broadcast to all GPRs. It is loaded into a register wich has its LSBs hardwired to the GPR number (0 thru 63). Then if GEN_INDEX is high, the mux selects the auto-count value and it is loaded into the GPRs to be either used to retrieve data using the TP or sent to the SX for the RB to use it to write the data to memory

Figure 12: GPR input mux Control

20. State management

Every clock, the sequencer will report to the CP the oldest states still in the pipe. These are the states of the programs as they enter the last ALU clause.

20.1 Parameter cache synchronization

In order for the sequencer not to begin a group of pixels before the associated group of vertices has finished, the sequencer will keep a 6 bit count per state (for a total of 8 counters). These counters are initialized to 0 and every time a vertex shader exports its data TO THE PARAMETER CACHE, the corresponding pointer is incremented. When the SC sends a new vector of pixels with the SC_SQ_new_vector bit asserted, the sequencer will first check if the count is greater than 0 before accepting the transmission (it will in fact accept the transmission but then lower its ready to receive). Then the sequencer waits for the count to go to one and decrements it. The sequencer can then issue the group of pixels to the interpolators. Every time the state changes, the new state counter is initialized to 0.

21. XY Address imports

The SC will be able to send the XY addresses to the GPRs. It does so by interleaving the writes of the IJs (to the IJ buffer) with XY writes (to the XY buffer). Then when writing the data to the GPRs, the sequencer is going to interpolate the IJ data or pass the XY data thru a Fix→float converter and expander and write the converted values to the GPRs. The Xys are currently SCREEN SPACE COORDINATES. The values in the XY buffers will wrap. See section 19.2 for details on how to control the interpolation in this mode.

21.1 Vertex indexes imports

In order to import vertex indexes, we have 16 8x96 staging registers. These are loaded one line at a time by the VGT block (96 bits). They are loaded in floating point format and can be transferred in 4 or 8 clocks to the GPRs.

22. Registers

22.1 Control

REG_DYNAMIC	Dynamic allocation (pixel/vertex) of the register file on or off.
REG_SIZE_PIX	Size of the register file's pixel portion (minimal size when dynamic allocation turned on)
REG_SIZE_VTX	Size of the register file's vertex portion (minimal size when dynamic allocation turned on)
ARBITRATION_POLICY	policy of the arbitration between vertexes and pixels
INST_BASE_VTX	start point for the vertex instruction store (RT always ends at vertex_base and Begins at 0)
INST_BASE_PIX	start point for the pixel shader instruction store
ONE_THREAD	debug state register. Only allows one program at a time into the GPRs
ONE_ALU	debug state register. Only allows one ALU program at a time to be executed (instead of 2)
INSTRUCTION	This is where the CP puts the base address of the instruction writes and type (auto-incremented on reads/writes) Register mapped
CONSTANTS	512*4 ALU constants + 32*6 Texture state 32 bits registers (logically mapped)
CONSTANTS_RT	256*4 ALU constants + 32*6 texture states? (physically mapped)
CONSTANT_EO_RT	This is the size of the space reserved for real time in the constant store (from 0 to CONSTANT_EO_RT). The re-mapping table operates on the rest of the memory

TSTATE_EO_RT This is the size of the space reserved for real time in the fetch state store (from 0 to TSTATE_EO_RT). The re-mapping table operates on the rest of the memory

22.2 Context

PS_BASE	base pointer for the pixel shader in the instruction store
VS_BASE	base pointer for the vertex shader in the instruction store
VS_CF_SIZE	size of the vertex shader (# of instructions in control program/2)
PS_CF_SIZE	size of the pixel shader (# of instructions in control program/2)
PS_SIZE	size of the pixel shader (cntl+instructions)
VS_SIZE	size of the vertex shader (cntl+instructions)
PS_NUM_REG	number of GPRs to allocate for pixel shader programs
VS_NUM_REG	number of GPRs to allocate for vertex shader programs
PARAM_SHADE	One 16 bit register specifying which parameters are to be gouraud shaded (0 = flat, 1 = gouraud)
PARAM_WRAP	64 bits: for which parameters (and channels (xyzw)) do we do the cyl wrapping (0=linear, 1=cylindrical).
PS_EXPORT_MODE	0xxxx : Normal mode

	1xxxx : Multipass mode
	If normal, bbbz where bbb is how many colors (0-4) and z is export z or not
	If multipass 1-12 exports for color.
VS_EXPORT_MODE	0: position (1 vector), 1: position (2 vectors), 3:multipass
VS_EXPORT_COUNT	Number of locations exported by the VS (and thus number of interpolated parameters)
PARAM_GEN_I0	Do we overwrite or not the parameter 0 with XY data and generated T and S values
GEN_INDEX	Auto generates an address from 0 to XX. Puts the results into R0-1 for pixel shaders and R2 for vertex shaders
CONST_BASE_VTX (9 bits)	Logical Base address for the constants of the Vertex shader
CONST_BASE_PIX (9 bits)	Logical Base address for the constants of the Pixel shader
CONST_SIZE_PIX (8 bits)	Size of the logical constant store for pixel shaders
CONST_SIZE_VTX (8 bits)	Size of the logical constant store for vertex shaders
INST_PRED_OPTIMIZE	Turns on the predicate bit optimization (if of, conditional_execute_predicates is always executed).
CF_BOOLEANS	256 boolean bits
CF_LOOP_COUNT	32x8 bit counters (number of times we traverse the loop)
CF_LOOP_START	32x8 bit counters (init value used in index computation)
CF_LOOP_STEP	32x8 bit counters (step value used in index computation)

23. DEBUG Registers

23.1 Context

DB_PROB_ADDR	instruction address where the first problem occurred
DB_PROB_COUNT	number of problems encountered during the execution of the program
DB_PROB_BREAK	break the clause if an error is found.
DB_ON	turns on an off debug method 2
DB_INST_COUNT	instruction counter for debug method 2
DB_BREAK_ADDR	break address for method number 2

23.2 Control

DB_ALUCST_MEMSIZE	Size of the physical ALU constant memory
DB_TSTATE_MEMSIZE	Size of the physical texture state memory

24. Interfaces

24.1 External Interfaces

Whenever an x is used, it means that the bus is broadcast to all units of the same name. For example, if a bus is named SQ→SPx it means that SQ is going to broadcast the same information to all SP instances.

24.2 SC to SP Interfaces

24.2.1 *SC_SP#*

There is one of these interfaces at front of each of the SP (buffer to stage pixel interpolators). This interface transmits the I,J data for pixel interpolation. For the entire system, two quads per clock are transferred to the 4 SPs, so each of these 4 interfaces transmits one half of a quad per clock. The interface below describes a half of a quad worth of data.

The actual data which is transferred per quad is

Ref Pix I => S4.20 Floating Point I value
Ref Pix J => S4.20 Floating Point J value

Delta Pix I (x3) => S4.8 Floating Point Delta I value
Delta Pix J (x3) => S4.8 Floating Point Delta J value
This equates to a total of 128 bits which transferred over 2 clocks
and therefor needs an interface 64 bits wide

Additionally, X,Y data (12-bit unsigned fixed) is conditionally sent across this data bus over the same wires in an additional clock. The X,Y data is sent on the lower 24 bits of the data bus with faceness in the msb.
Transfers across these interfaces are synchronized with the SC_SQ IJ Control Bus transfers.

The data transfer across each of these busses is controlled by a IJ_BUF_INUSE_COUNT in the SC. Each time the SC has sent a pixel vector's worth of data to the SPs, he will increment the IJ_BUF_INUSE_COUNT count. Prior to sending the next pixel vectors data, he will check to make sure the count is less than MAX_BUFER_MINUS_2, if not the SC will stall until the SQ returns a pipelined pulse to decrement the count when he has scheduled a buffer free. Note: We could/may optimize for the case of only sending only IJ to use all the buffers to pre-load more. Currently it is planned for the SP to hold 2 double buffers of I,J data and two buffers of X,Y data, so if either X,Y or Centers and Centroids are on, then the SC can send two Buffers.

In at least the initial version, the SC shall send 16 quads per pixel vector even if the vector is not full. This will increment buffer write address pointers correctly all the time. (We may revisit this for both the SX,SP,SQ and add a EndOfVector signal on all interfaces to quit early. We opted for the simple mode first with a belief that only the end of packet and multiple new vector signals should cause a partial vector and that this would not really be significant performance hit.)

Name	Bits	Description
SC_SP#_data	64	IJ information sent over 2 clocks (or X,Y in 24 LSBs with faceness in upper bit) **Type 0 or 1**, First clock I, second clk J Field ULC URC LLC LRC Bits [63:39] [38:26] [25:13] [12:0] Format SE4M20 SE4M8 SE4M8 SE4M8 **Type 2** Field Face X Y Bits [63] [23:12] [11:0] Format Bit Unsigned Unsigned
SC_SP#_valid	1	Valid
SC_SP#_last_quad_data	1	This bit will be set on the last transfer of data per quad.
SC_SP#_type	2	0 -> Indicates centroids 1 -> Indicates centers 2 -> Indicates X,Y Data and faceness on data bus The SC shall look at state data to determine how many types to send for the interpolation process.

The # is included for clarity in the spec and will be replaced with a prefix of u#_ in the verilog module statement for the SC and the SP block will have neither because the instantiation will insert the prefix.

24.2.2 *SC_SQ*

This is the control information sent to the sequencer in order to synchronize and control the interpolation and/or loading data into the GPRs needed to execute a shader program on the sent pixels. This data will be sent over two clocks per transfer with 1 to 16 transfers. Therefore the bus (approx 92 bits) could be folded in half to approx 47 bits.

Name	Bits	Description
SC_SQ_data	46	Control Data sent to the SQ 1 clk transfers Event – valid data consist of event_id and state_id. Instruct SQ to post an event vector to send state id and event_id through request fifo and onto the reservation stations

		making sure state id and/or event_id gets back to the CP. Events only follow end of packets so no pixel vectors will be in progress. Empty Quad Mask – Transfer Control data consisting of pc_dealloc or new_vector. Receipt of this is to transfer pc_dealloc or new_vector without any valid quad data. New vector will always be posted to request fifo and pc_dealloc will be attached to any pixel vector outstanding or posted in request fifo if no valid quad outstanding. 2 clk transfers Quad Data Valid – Sending quad data with or without new_vector or pc_dealloc. New vector will be posted to request fifo with or without a pixel vector and pc_dealloc will be posted with a pixel vector unless none is in progress. In this case the pc_dealloc will be posted in the request queue. Filler quads will be transferred with The Quad mask set but the pixel corresponding pixel mask set to zero.
SC_SQ_valid	1	SC sending valid data, 2nd clk could be all zeroes

SC_SQ_data – first clock and second clock transfers are shown in the table below.

Name	**BitField**	**Bits**	**Description**
1st Clock Transfer			
SC_SQ_event	0	1	This transfer is a 1 clock event vector Force quad_mask = new_vector=pc_dealloc=0
SC_SQ_event_id	[2:1]	2	This field identifies the event 0 => denotes an End Of State Event 1 => TBD
SC_SQ_pc_dealloc	[5:3]	3	Deallocation token for the Parameter Cache
SC_SQ_new_vector	6	1	The SQ must wait for Vertex shader done count > 0 and after dispatching the Pixel Vector the SQ will decrement the count.
SC_SQ_quad_mask	[10:7]	4	Quad Write mask left to right SP0 => SP3
SC_SQ_end_of_prim	11	1	End Of the primitive
SC_SQ_state_id	[14:12]	3	State/constant pointer (6*3+3)
SC_SQ_pix_mask	[30:15]	16	Valid bits for all pixels SP0=>SP3 (UL,UR,LL,LR)
SC_SQ_prim_type	[33:31]	3	Stippled line and Real time command need to load tex cords from alternate buffer 000: Normal 010: Realtime 101: Line AA 110: Point AA (Sprite)
SC_SQ_provok_vtx	[35:34]	2	Provoking vertex for flat shading
SC_SQ_pc_ptr0	[46:36]	11	Parameter Cache pointer for vertex 0
2nd Clock Transfer			
SC_SQ_pc_ptr1	[10:0]	11	Parameter Cache pointer for vertex 1

SC_SQ_pc_ptr2	[21:11]	11	Parameter Cache pointer for vertex 2
SC_SQ_lod_correct	[45:22]	24	LOD correction per quad (6 bits per quad)

Name	Bits	Description
SQ_SC_free_buff	1	Pipelined bit that instructs SC to decrement count of buffers in use.
SQ_SC_dec_cntr_cnt	1	Pipelined bit that instructs SC to decrement count of new vector and/or event sent to prevent SC from overflowing SQ interpolator/Reservation request fifo.

The scan converter will submit a partial vector whenever:

1.) He gets a primitive marked with an end of packet signal.

2.) A current pixel vector is being assembled with at least one or more valid quads and the vector has been marked for deallocate when a primitive marked new_vector arrives. The Scan Converter will submit a partial vector (up to 16quads with zero pixel mask to fill out the vector) prior to submitting the new_vector marker\primitive.

(This will prevent a hang which can be demonstrated when all primitives in a packet three vectors are culled except for a one quad primitive that gets marked pc_dealloc (vertices maximum size). In this case two new_vectors are submitted and processed, but then one valid quad with the pc_dealloc creates a vector and then the new would wait for another vertex vector to be processed, but the one being waited for could never export until the pc_dealloc signal made it through and thus the hang.)

24.2.3 *SQ to SX: Interpolator bus*

Name	Direction	Bits	Description
SQ_SXx_interp_flat_vtx	SQ→SPx	2	Provoking vertex for flat shading
SQ_SXx_interp_flat_gouraud	SQ→SPx	1	Flat or gouraud shading
SQ_SXx_interp_cyl_wrap	SQ→SPx	4	Wich channel needs to be cylindrical wrapped
SQ_SXx_pc_ptr0	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_pc_ptr1	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_pc_ptr2	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_rt_sel	SQ→SXx	1	Selects between RT and Normal data
SQ_SXx_pc_wr_en	SQ→SXx	1	Write enable for the PC memories
SQ_SXx_pc_wr_addr	SQ→SXx	7	Write address for the PCs
SQ_SXx_pc_channel_mask	SQ→SXx	4	Channel mask

24.2.4 *SQ to SP: Staging Register Data*

This is a broadcast bus that sends the VSISR information to the staging registers of the shader pipes.

Name	Direction	Bits	Description
SQ_SPx_vsr_data	SQ→SPx	96	Pointers of indexes or HOS surface information
SQ_SPx_vsr_double	SQ→SPx	1	0: Normal 96 bits per vert 1: double 192 bits per vert
SQ_SP0_vsr_valid	SQ→SP0	1	Data is valid
SQ_SP1_ vsr_ valid	SQ→SP1	1	Data is valid
SQ_SP2_ vsr_ valid	SQ→SP2	1	Data is valid
SQ_SP3_ vsr_ valid	SQ→SP3	1	Data is valid
SQ_SPx_vsr_read	SQ→SPx	1	Increment the read pointers

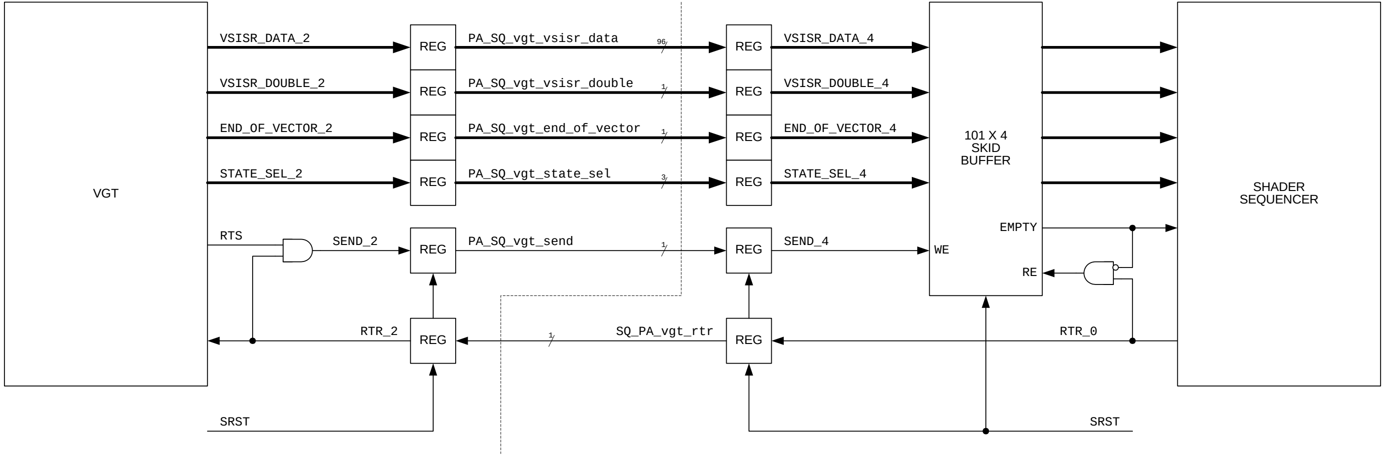
24.2.5 *VGT to SQ : Vertex interface*

24.2.5.1 Interface Signal Table

The area difference between the two methods is not sufficient to warrant complicating the interface or the state requirements of the VSISRs. **Therefore, the POR for this interface is that the VGT will transmit the data to the VSISRs (via the Shader Sequencer) in full, 32-bit floating-point format.** The VGT can transmit up to six 32-bit floating-point values to each VSISR where four or more values require two transmission clocks. The data bus is 96 bits wide.

Name	Bits	Description
VGT_SQ_vsisr_data	96	Pointers of indexes or HOS surface information
VGT_SQ_vsisr_double	1	0: Normal 96 bits per vert 1: double 192 bits per vert
VGT_SQ_end_of_vector	1	Indicates the last VSISR data set for the current process vector (for double vector data, "end_of_vector" is set on the first vector)
VGT_SQ_indx_valid	1	Vsisr data is valid
VGT_SQ_state	3	Render State (6*3+3 for constants). This signal is guaranteed to be correct when “VGT_SQ_vgt_end_of_vector” is high.
VGT_SQ_send	1	Data on the VGT_SQ is valid receive (see write-up for standard R400 SEND/RTR interface handshaking)
SQ_VGT_rtr	1	Ready to receive (see write-up for standard R400 SEND/RTR interface handshaking)

24.2.5.2 Interface Diagrams



VGT
VSISR_DATA_2
VSISR_DOUBLE_2
END_OF_VECTOR_2
STATE_SEL_2
RTS
SEND_2
RTR_2
SRST
REG
PA_SQ_vgt_vsisr_data
96
PA_SQ_vgt_vsisr_double
1
PA_SQ_vgt_end_of_vector
1
PA_SQ_vgt_state_sel
3
PA_SQ_vgt_send
1
SQ_PA_vgt_rtr
1
VSISR_DATA_4
VSISR_DOUBLE_4
END_OF_VECTOR_4
STATE_SEL_4
SEND_4
101 X 4
SKID
BUFFER
EMPTY
WE
RE
RTR_0
SRST
SHADER
SEQUENCER

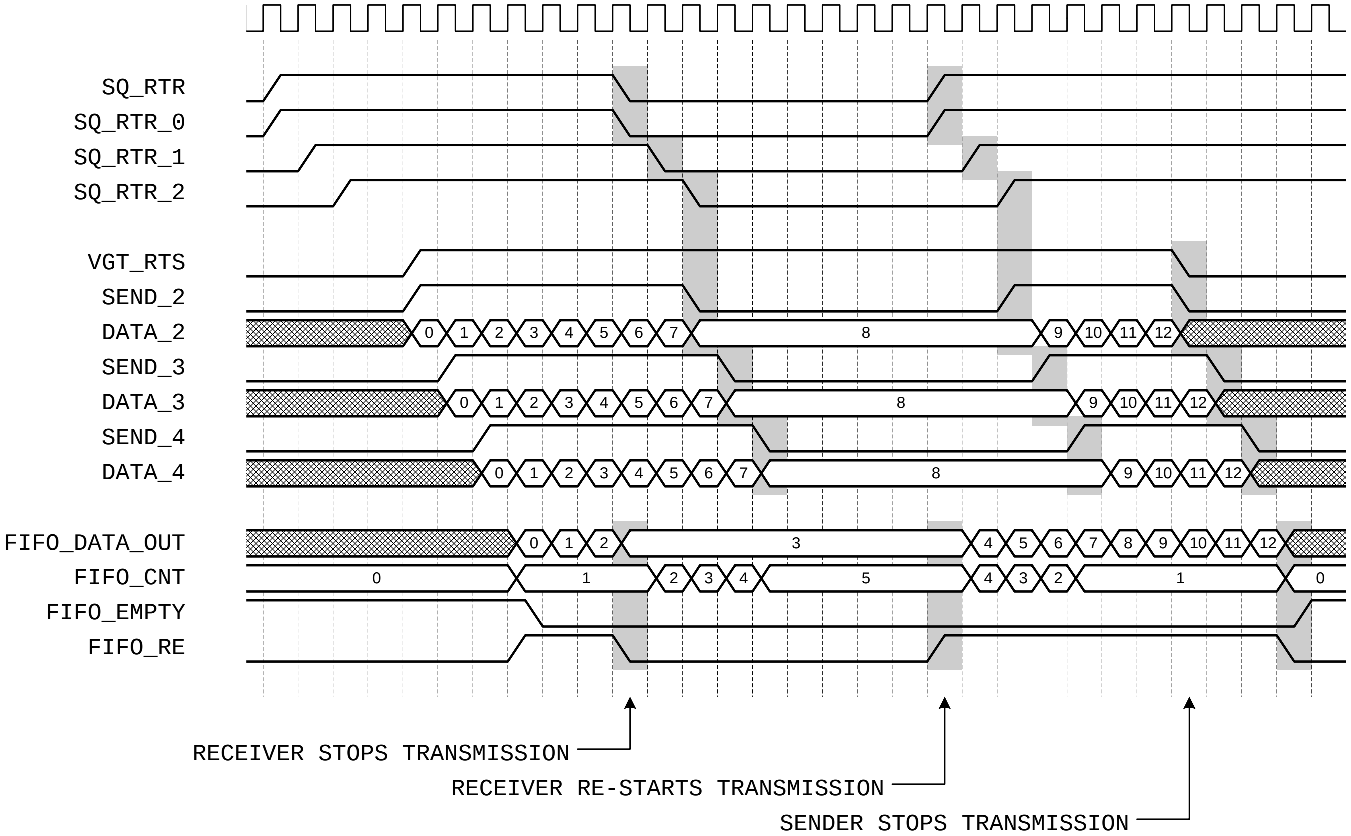


Figure 1. Detailed Logical Diagram for PA_SQ_vgt Interface.

24.2.6 *SQ to SX: Control bus*

Name	Direction	Bits	Description
SQ_SXx_exp_type	SQ→SXx	2	00: Pixel without z (1 to 4 buffers) 01: Pixel with z (1 to 4 buffers) 10: Position (1 or 2 results) 11: Pass thru (4,8 or 12 results aligned)
SQ_SXx_exp_number	SQ→SXx	2	Number of locations needed in the export buffer (encoding depends on the type see bellow).
SQ_SXx_exp_alu_id	SQ→SXx	1	ALU ID
SQ_SXx_exp_valid	SQ→SXx	1	Valid bit
SQ_SXx_exp_state	SQ→SXx	3	State Context
SQ_SXx_free_done	SQ→SXx	1	Pulse to indicate that the previous export is finished (this can be sent with or without the other fields of the interface)
SQ_SXx_free_alu_id	SQ→SXx	1	ALU ID

Depending on the type the number of export location changes:

- Type 00 : Pixels without Z
  - 00 = 1 buffer
  - 01 = 2 buffers
  - 10 = 3 buffers
  - 11 = 4 buffer
- Type 01: Pixels with Z
  - 00 = 2 Buffers (color + Z)
  - 01 = 3 buffers (2 color + Z)
  - 10 = 4 buffers (3 color + Z)
  - 11 = 5 buffers (4 color + Z)
- Type 10 : Position export
  - 00 = 1 position
  - 01 = 2 positions
  - 1X = Undefined
- Type 11: Pass Thru
  - 00 = 4 buffers
  - 01 = 8 buffers
  - 10 = 12 buffers
  - 11 = Undefined

Below the thick black line is the end of transfer packet that tells the SX that a given export is finished. The report packet **will always arrive either before or at the same time than the next export to the same ALU id**.

24.2.7 *SX to SQ : Output file control*

Name	Direction	Bits	Description
SXx_SQ_exp_count_rdy	SXx→SQ	1	Raised by SX0 to indicate that the following two fields reflect the result of the most recent export
SXx_SQ_exp_pos_avail	SXx→SQ	1	Specifies whether there is room for another position.
SXx_SQ_exp_buf_avail	SXx→SQ	7	Specifies the space available in the output buffers. 0: buffers are full 1: 2K-bits available (32-bits for each of the 64 pixels in a clause) ... 64: 128K-bits available (16 128-bit entries for each of 64 pixels) 65-127: RESERVED

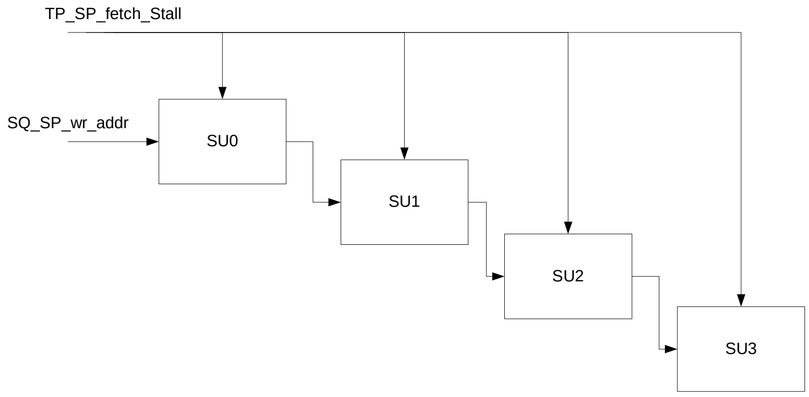
24.2.8 SQ to TP: Control bus

Once every clock, the fetch unit sends to the sequencer on which RS line it is now working and if the data in the GPRs is ready or not. This way the sequencer can update the fetch valid bits flags for the reservation station. The sequencer also provides the instruction and constants for the fetch to execute and the address in the register file where to write the fetch return data.

Name	Direction	Bits	Description
TPx_SQ_data_rdy	TPx→ SQ	1	Data ready
TPx_SQ_rs_line_num	TPx→ SQ	6	Line number in the Reservation station
TPx_SQ_type	TPx→ SQ	1	Type of data sent (0:PIXEL, 1:VERTEX)
SQ_TPx_send	SQ→TPx	1	Sending valid data
SQ_TPx_const	SQ→TPx	48	Fetch state sent over 4 clocks (192 bits total)
SQ_TPx_instr	SQ→TPx	24	Fetch instruction sent over 4 clocks
SQ_TPx_end_of_group	SQ→TPx	1	Last instruction of the group
SQ_TPx_Type	SQ→TPx	1	Type of data sent (0:PIXEL, 1:VERTEX)
SQ_TPx_gpr_phase	SQ→TPx	2	Write phase signal
SQ_TP0_lod_correct	SQ→TP0	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP0_pix_mask	SQ→TP0	4	Pixel mask 1 bit per pixel
SQ_TP1_lod_correct	SQ→TP1	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP1_pix_mask	SQ→TP1	4	Pixel mask 1 bit per pixel
SQ_TP2_lod_correct	SQ→TP2	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP2_pix_mask	SQ→TP2	4	Pixel mask 1 bit per pixel
SQ_TP3_lod_correct	SQ→TP3	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP3_pix_mask	SQ→TP3	4	Pixel mask 1 bit per pixel
SQ_TPx_rs_line_num	SQ→TPx	6	Line number in the Reservation station
SQ_TPx_write_gpr_index	SQ->TPx	7	Index into Register file for write of returned Fetch Data

24.2.9 TP to SQ: Texture stall

The TP sends this signal to the SQ and the SPs when its input buffer is full.



Name	Direction	Bits	Description
TP_SQ_fetch_stall	TP→ SQ	1	Do not send more texture request if asserted

24.2.10 *SQ to SP: Texture stall*

Name	Direction	Bits	Description
SQ_SPx_fetch_stall	SQ→SPx	1	Do not send more texture request if asserted

24.2.11 *SQ to SP: GPR and auto counter*

Name	Direction	Bits	Description
SQ_SPx_gpr_wr_addr	SQ→SPx	7	Write address
SQ_SPx_gpr_rd_addr	SQ→SPx	7	Read address
SQ_SPx_gpr_rd_en	SQ→SPx	1	Read Enable
SQ_SP0_gpr_wr_en	SQ→SPx	1	Write Enable for the GPRs of SP0
SQ_SP1_gpr_wr_en	SQ→SPx	1	Write Enable for the GPRs of SP1
SQ_SP2_gpr_wr_en	SQ→SPx	1	Write Enable for the GPRs of SP2
SQ_SP~~x~~3_gpr_wr_en	SQ→SPx	1	Write Enable for the GPRs of SP3
SQ_SPx_gpr_phase	SQ→SPx	2	The phase mux (arbitrates between inputs, ALU SRC reads and writes)
SQ_SPx_channel_mask	SQ→SPx	4	The channel mask
SQ_SPx_gpr_input_sel	SQ→SPx	2	When the phase mux selects the inputs this tells from which source to read from: Interpolated data, VTX0, VTX1, autogen counter.
SQ_SPx_auto_count	SQ→SPx	12?	Auto count generated by the SQ, common for all shader pipes

24.2.12 *SQ to SPx: Instructions*

Name	Direction	Bits	Description
SQ_SPx_instr_start	SQ→SPx	1	Instruction start
SQ_SP_instr	SQ→SPx	21	Transferred over 4 cycles 0: SRC A Select 2:0 SRC A Argument Modifier 3:3 SRC A swizzle 11:4 VectorDst 17:12 Unused 20:18 --- - 1: SRC B Select 2:0 SRC B Argument Modifier 3:3 SRC B swizzle 11:4 ScalarDst 17:12 Unused 20:18 --- - 2: SRC C Select 2:0 SRC C Argument Modifier 3:3 SRC C swizzle 11:4 Unused 20:12 --- - 3: Vector Opcode 4:0 Scalar Opcode 10:5 Vector Clamp 11:11 Scalar Clamp 12:12 Vector Write Mask 16:13 Scalar Write Mask 20:17
SQ_SPx_exp_alu_id	SQ→SPx	1	ALU ID
SQ_SPx_exporting	SQ→SPx	2	0: Not Exporting 1: Vector Exporting 2: Scalar Exporting
SQ_SPx_stall	SQ→SPx	1	Stall signal
SQ_SP0_write_mask	SQ→SP0	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP1_ write_mask	SQ→SP1	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP2_ write_mask	SQ→SP2	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP3_ write_mask	SQ→SP3	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock

24.2.13 *SP to SQ: Constant address load/ Predicate Set*

Name	Direction	Bits	Description
SP0_SQ_const_addr	SP0→SQ	36	Constant address load / predicate vector load (4 bits only) to the sequencer
SP0_SQ_valid	SP0→SQ	1	Data valid
SP1_SQ_const_addr	SP1→SQ	36	Constant address load / predicate vector load (4 bits only) to the sequencer

SP1_SQ_valid	SP1→SQ	1	Data valid
SP2_SQ_const_addr	SP2→SQ	36	Constant address load / predicate vector load (4 bits only) to the sequencer
SP2_SQ_valid	SP2→SQ	1	Data valid
SP3_SQ_const_addr	SP3→SQ	36	Constant address load / predicate vector load (4 bits only) to the sequencer
SP3_SQ_valid	SP3→SQ	1	Data valid

24.2.14 *SQ to SPx: constant broadcast*

Name	Direction	Bits	Description
SQ_SPx_const	SQ→SPx	128	Constant broadcast

24.2.15 *SP0 to SQ: Kill vector load*

Name	Direction	Bits	Description
SP0_SQ_kill_vect	SP0→SQ	4	Kill vector load
SP1_SQ_kill_vect	SP1→SQ	4	Kill vector load
SP2_SQ_kill_vect	SP2→SQ	4	Kill vector load
SP3_SQ_kill_vect	SP3→SQ	4	Kill vector load

24.2.16 *SQ to CP: RBBM bus*

Name	Direction	Bits	Description
SQ_RBB_rs	SQ→CP	1	Read Strobe
SQ_RBB_rd	SQ→CP	32	Read Data
SQ_RBBM_nrtrtr	SQ→CP	1	Optional
SQ_RBBM_rtr	SQ→CP	1	Real-Time (Optional)

24.2.17 *CP to SQ: RBBM bus*

Name	Direction	Bits	Description
rbbm_we	CP→SQ	1	Write Enable
rbbm_a	CP→SQ	15	Address -- Upper Extent is TBD (16:2)
rbbm_wd	CP→SQ	32	Data
rbbm_be	CP→SQ	4	Byte Enables
rbbm_re	CP→SQ	1	Read Enable
rbb_rs0	CP→SQ	1	Read Return Strobe 0
rbb_rs1	CP→SQ	1	Read Return Strobe 1
rbb_rd0	CP→SQ	32	Read Data 0
rbb_rd1	CP→SQ	32	Read Data 0
RBBM_SQ_soft_reset	CP→SQ	1	Soft Reset

24.2.18 *SQ to CP: State report*

Name	Direction	Bits	Description
SQ_CP_vs_event	SQ→CP	1	Vertex Shader Event
SQ_CP_vs_eventid	SQ→CP	2	Vertex Shader Event ID
SQ_CP_ps_event	SQ→CP	1	Pixel Shader Event
SQ_CP_ps_eventid	SQ→CP	2	Pixel Shader Event ID

eventid = 0 => *sEndOfState (i.e. VsEndOfState)
eventid = 1 => *sDone (i.e. VsDone)

So, the CP will assume the Vs is done with a state whenever it gets a pulse on the SQ_CP_vs_event and the SQ_CP_vs_eventid = 0.

24.3 Example of control flow program execution

We now provide some examples of execution to better illustrate the new design.

Given the program:

Alu 0
Alu 1
Tex 0
Tex 1
Alu 3 Serial
Alu 4
Tex 2
Alu 5
Alu 6 Serial
Tex 3
Alu 7
Alloc Position 1 buffer
Alu 8 Export
Tex 4
Alloc Parameter 3 buffers
Alu 9 Export 0
Tex 5
Alu 10 Serial Export 2
Alu 11 Export 1 End

Would be converted into the following CF instructions:

```
Execute Alu 0 Alu 0 Tex 0 Tex 0 Alu 1 Alu 0 Tex 0 Alu 0 Alu 1 Tex 0
Execute Alu 0
Alloc Position 1
Execute Alu 0 Tex 0
Alloc Param 3
Execute Alu 0 Tex 0 Alu 1 Alu 0 End
```

And the execution of this program would look like this:

Put thread in Vertex RS:

Control Flow Instruction Pointer (12 bits), (CFP)
Execution Count Marker (3 or 4 bits), (ECM)
Loop Iterators (4x9 bits), (LI)
Call return pointers (4x12 bits), (CRP)
Predicate Bits(4x64 bits), (PB)
Export ID (1 bit), (EXID)
GPR Base Ptr (8 bits), (GPR)
Export Base Ptr (7 bits), (EB)
Context Ptr (3 bits).(CPTR)
LOD correction bits (16x6 bits) (LOD)

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	0	0	0	0	0	0	0	0	0

Valid Thread (VALID)
Texture/ALU engine needed (TYPE)
Texture Reads are outstanding (PENDING)
Waiting on Texture Read to Complete (SERIAL)
Allocation Wait (2 bits) (ALLOC)

00 – No allocation needed
01 – Position export allocation needed (ordered export)
10 – Parameter or pixel export needed (ordered export)
11 – pass thru (out of order export)
Allocation Size (4 bits) (SIZE)
Position Allocated (POS_ALLOC)
First thread of a new context (FIRST)
Last (1 bit), (LAST)

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	0	0	0	0	0	1	0

Then the thread is picked up for the execution of the first control flow instruction:

```
Execute Alu 0 Alu 0 Tex 0 Tex 0 Alu 1 Alu 0 Tex 0 Alu 0 Alu 1 Tex 0
```

It executes the first two ALU instructions and goes back to the RS for a resource request change. Here is the state returned to the RS:

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	2	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	0	0	0	0	0	1	0

Then when the texture pipe frees up, the arbiter picks up the thread to issue the texture reads. The thread comes back in this state:

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	4	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	1	0	0	0	1	0

Because of the serial bit the arbiter must wait for the texture to return and clear the PENDING bit before it can pick the thread up. Lets say that the texture reads are complete, then the arbiter picks up the thread and returns it in this state:

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	6	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	0	0	0	0	0	1	0

Again the TP frees up, the arbiter picks up the thread and executes. It returns in this state:

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	7	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	0	0	0	1	0

Now, even if the texture has not returned we can still pick up the thread for ALU execution because the serial bit is not set. The thread will however come back to the RS for the second ALU instruction because it has the serial bit set.

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	8	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	1	0	0	0	1	0

As soon as the TP clears the pending bit the thread is picked up and returns:

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	9	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	0	0	0	0	0	1	0

Picked up by the TP and returns:

```
Execute Alu 0
```

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
1	0	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	0	0	0	1	0

Picked up by the ALU and returns (lets say the TP has not returned yet):

```
Alloc Position 1
```

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
2	0	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	01	1	0	1	0

If the SX has the place for the export, the SQ is going to allocate and pick up the thread for execution. It returns to the RS in this state:

```
Execute Alu 0 Tex 0
```

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
3	1	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	1	0	0	0	1	1	0

Now, since the TP has not returned yet, we must wait for it to return because we cannot issue multiple texture requests. The TP returns, clears the PENDING bit and we proceed:

```
    Alloc Param 3
```

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
4	0	0	0	0	1	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	10	3	1	1	0

Once again the SQ makes sure the SX has enough room in the Parameter cache before it can pick up this thread.

```
    Execute Alu 0 Tex 0 Alu 1 Alu 0 End
```

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
5	1	0	0	0	1	0	100	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	1	0	0	0	1	1	0

This executes on the TP and then returns:

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
5	2	0	0	0	1	0	100	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	1	0	0	1	1	1

Waits for the TP to return because of the textures reads are pending (and SERIAL in this case). Then executes and does not return to the RS because the LAST bit is set. This is the end of this thread and before dropping it on the floor, the SQ notifies the SX of export completion.

25. Open issues

Need to do some testing on the size of the register file as well as on the register file allocation method (dynamic VS static).

Saving power?

	ORIGINATE DATE 24 September, 2001	EDIT DATE 4 September, 2015~~13~~ May, 20022 May, 2002	DOCUMENT-REV. NUM. GEN-CXXXXX-REVA	PAGE 1 of 53

Author: Laurent Lefebvre

Issue To:	**Copy No:**

R400 Sequencer Specification

SQ

Version 2.02~~1~~

Overview: This is an architectural specification for the R400 Sequencer block (SEQ). It provides an overview of the required capabilities and expected uses of the block. It also describes the block interfaces, internal sub-blocks, and provides internal state diagrams.

AUTOMATICALLY UPDATED FIELDS:

Document Location: C:\perforce\r400\doc_lib\design\blocks\sq\R400_Sequencer.doc

Current Intranet Search Title: R400 Sequencer Specification

APPROVALS	
Name/Dept	Signature/Date

Remarks:

THIS DOCUMENT CONTAINS CONFIDENTIAL INFORMATION THAT COULD BE SUBSTANTIALLY DETRIMENTAL TO THE INTEREST OF ATI TECHNOLOGIES INC. THROUGH UNAUTHORIZED USE OR DISCLOSURE.

Table Of Contents

1. OVERVIEW **7**
1.1 Top Level Block Diagram 9
1.2 Data Flow graph (SP) 10
1.3 Control Graph 11
2. INTERPOLATED DATA BUS **11**
3. INSTRUCTION STORE **14**
4. SEQUENCER INSTRUCTIONS **14**
5. CONSTANT STORES **14**
5.1 Memory organizations 14
5.2 Management of the Control Flow Constants 15
5.3 Management of the re-mapping tables 15
5.3.1 R400 Constant management 15
5.3.2 Proposal for R400LE constant management 15
5.3.3 Dirty bits 17
5.3.4 Free List Block 17
5.3.5 De-allocate Block 18
5.3.6 Operation of Incremental model 18
5.4 Constant Store Indexing 18
5.5 Real Time Commands 19
5.6 Constant Waterfalling 19
6. LOOPING AND BRANCHES **20**
6.1 The controlling state. 20
6.2 The Control Flow Program 20
6.2.1 Control flow instructions table 21
6.3 Implementation ..... 23~~22~~
6.4 Data dependant predicate instructions ..... 25~~24~~
6.5 HW Detection of PV,PS ..... 25~~24~~
6.6 Register file indexing ..... 25~~2524~~
6.7 Debugging the Shaders ..... 26~~25~~
6.7.1 Method 1: Debugging registers ..... 26~~25~~
6.7.2 Method 2: Exporting the values in the GPRs ..... 26~~2625~~
7. PIXEL KILL MASK **26**
8. MULTIPASS VERTEX SHADERS (HOS) ..... **27~~26~~**
9. REGISTER FILE ALLOCATION ..... **27~~26~~**
10. FETCH ARBITRATION ..... **28~~27~~**
11. ALU ARBITRATION ..... **28~~27~~**
12. HANDLING STALLS ..... **29~~28~~**
13. CONTENT OF THE RESERVATION STATION FIFOS ..... **29~~28~~**
14. THE OUTPUT FILE ..... **29~~28~~**
15. IJ FORMAT ..... **29~~28~~**
15.1 Interpolation of constant attributes ..... 30~~29~~
16. STAGING REGISTERS ..... **30~~29~~**

17. THE PARAMETER CACHE .... **32~~3130~~**
17.1 Export restrictions .... 33~~3130~~
17.1.1 Pixel exports: .... 33~~3130~~
17.1.2 Vertex exports: .... 33~~3130~~
17.1.3 Pass thru exports: .... 33~~3130~~
17.2 Arbitration restrictions .... 33~~3130~~
18. EXPORT TYPES .... **33~~3231~~**
18.1 Vertex Shading .... 33~~3231~~
18.2 Pixel Shading .... 34~~3231~~
19. SPECIAL INTERPOLATION MODES .... **34~~3231~~**
19.1 Real time commands .... 34~~3231~~
19.2 Sprites/ XY screen coordinates/ FB information .... 34~~3332~~
19.3 Auto generated counters .... 35~~3332~~
19.3.1 Vertex shaders .... 35~~3332~~
19.3.2 Pixel shaders .... 35~~3332~~
20. STATE MANAGEMENT .... **35~~3433~~**
20.1 Parameter cache synchronization .... 35~~3433~~
21. XY ADDRESS IMPORTS .... **36~~3433~~**
21.1 Vertex indexes imports .... 36~~3433~~
22. REGISTERS .... **36~~3534~~**
22.1 Control .... 36~~3534~~
22.2 Context .... 36~~3534~~
23. DEBUG REGISTERS .... **37~~3635~~**
23.1 Context .... 37~~3635~~
23.2 Control .... 37~~3635~~
24. INTERFACES .... **37~~3635~~**
24.1 External Interfaces .... 37~~3635~~
24.2 SC to SP Interfaces .... 37~~3635~~
24.2.1 SC_SP# .... 37~~3635~~
24.2.2 SC_SQ .... 38~~3736~~
24.2.3 SQ to SX: Interpolator bus .... 40~~3938~~
24.2.4 SQ to SP: Staging Register Data .... 40~~3938~~
24.2.5 VGT to SQ : Vertex interface .... 40~~3938~~
24.2.6 SQ to SX: Control bus .... 44~~4241~~
24.2.7 SX to SQ : Output file control .... 44~~4241~~
24.2.8 SQ to TP: Control bus .... 45~~4342~~
24.2.9 TP to SQ: Texture stall .... 45~~4342~~
24.2.10 SQ to SP: Texture stall .... 46~~4442~~
24.2.11 SQ to SP: GPR and auto counter .... 46~~4443~~
24.2.12 SQ to SPx: Instructions .... 47~~4544~~
24.2.13 SP to SQ: Constant address load/ Predicate Set .... 47~~4544~~
24.2.14 SQ to SPx: constant broadcast .... 48~~4645~~

24.2.15 SP0 to SQ: Kill vector load .... 48~~4645~~
24.2.16 SQ to CP: RBBM bus .... 48~~4645~~
24.2.17 CP to SQ: RBBM bus .... 48~~4645~~
24.2.18 SQ to CP: State report .... 48~~4645~~
24.3 Example of control flow program execution .... 49~~4645~~
25. OPEN ISSUES .... 53~~5150~~

Revision Changes:

Rev 0.1 (Laurent Lefebvre) Date: May 7, 2001	First draft.
Rev 0.2 (Laurent Lefebvre) Date : July 9, 2001	Changed the interfaces to reflect the changes in the SP. Added some details in the arbitration section.
Rev 0.3 (Laurent Lefebvre) Date : August 6, 2001	Reviewed the Sequencer spec after the meeting on August 3, 2001.
Rev 0.4 (Laurent Lefebvre) Date : August 24, 2001	Added the dynamic allocation method for register file and an example (written in part by Vic) of the flow of pixels/vertices in the sequencer.
Rev 0.5 (Laurent Lefebvre) Date : September 7, 2001	Added timing diagrams (Vic)
Rev 0.6 (Laurent Lefebvre) Date : September 24, 2001	Changed the spec to reflect the new R400 architecture. Added interfaces.
Rev 0.7 (Laurent Lefebvre) Date : October 5, 2001	Added constant store management, instruction store management, control flow management and data dependant predication.
Rev 0.8 (Laurent Lefebvre) Date : October 8, 2001	Changed the control flow method to be more flexible. Also updated the external interfaces.
Rev 0.9 (Laurent Lefebvre) Date : October 17, 2001	Incorporated changes made in the 10/18/01 control flow meeting. Added a NOP instruction, removed the conditional_execute_or_jump. Added debug registers.
Rev 1.0 (Laurent Lefebvre) Date : October 19, 2001	Refined interfaces to RB. Added state registers.
Rev 1.1 (Laurent Lefebvre) Date : October 26, 2001	Added SEQ→SP0 interfaces. Changed delta precision. Changed VGT→SP0 interface. Debug Methods added.
Rev 1.2 (Laurent Lefebvre) Date : November 16, 2001	Interfaces greatly refined. Cleaned up the spec.
Rev 1.3 (Laurent Lefebvre) Date : November 26, 2001	Added the different interpolation modes.
Rev 1.4 (Laurent Lefebvre) Date : December 6, 2001	Added the auto incrementing counters. Changed the VGT→SQ interface. Added content on constant management. Updated GPRs.
Rev 1.5 (Laurent Lefebvre) Date : December 11, 2001	Removed from the spec all interfaces that weren't directly tied to the SQ. Added explanations on constant management. Added PA→SQ synchronization fields and explanation.
Rev 1.6 (Laurent Lefebvre) Date : January 7, 2002	Added more details on the staging register. Added detail about the parameter caches. Changed the call instruction to a Conditionnal_call instruction. Added details on constant management and updated the diagram.
Rev 1.7 (Laurent Lefebvre) Date : February 4, 2002	Added Real Time parameter control in the SX interface. Updated the control flow section.
Rev 1.8 (Laurent Lefebvre) Date : March 4, 2002	New interfaces to the SX block. Added the end of clause modifier, removed the end of clause instructions.
Rev 1.9 (Laurent Lefebvre) Date : March 18, 2002	Rearangement of the CF instruction bits in order to ensure byte alignement.
Rev 1.10 (Laurent Lefebvre) Date : March 25, 2002	Updated the interfaces and added a section on exporting rules.
Rev 1.11 (Laurent Lefebvre) Date : April 19, 2002	Added CP state report interface. Last version of the spec with the old control flow scheme
Rev 2.0 (Laurent Lefebvre) Date : April 19, 2002	New control flow scheme

Rev 2.01 (Laurent Lefebvre) Date : May 2, 2002	Changed slightly the control flow instructions to allow force jumps and calls.
Rev 2.02 (Laurent Lefebvre) Date : May 13, 2002	Updated the Opcodes. Added type field to the constant/pred interface. Added Last field to the SQ→SP instruction load interface.

1. Overview

The sequencer chooses two ALU threads and a fetch hread to execute, and executes all of the instructions in a block before looking for a new clause of the same type. Two ALU threads are executed interleaved to hide the ALU latency. The arbitrator will give priority to older threads. There are two separate reservation stations, one for pixel vectors and one for vertices vectors. This way a pixel can pass a vertex and a vertex can pass a pixel.

To support the shader pipe the sequencer also contains the shader instruction cache, constant store, control flow constants and texture state. The four shader pipes also execute the same instruction thus there is only one sequencer for the whole chip.

The sequencer first arbitrates between vectors of 64 vertices that arrive directly from primitive assembly and vectors of 16 quads (64 pixels) that are generated in the scan converter.

The vertex or pixel program specifies how many GPRs it needs to execute. The sequencer will not start the next vector until the needed space is available in the GPRs.

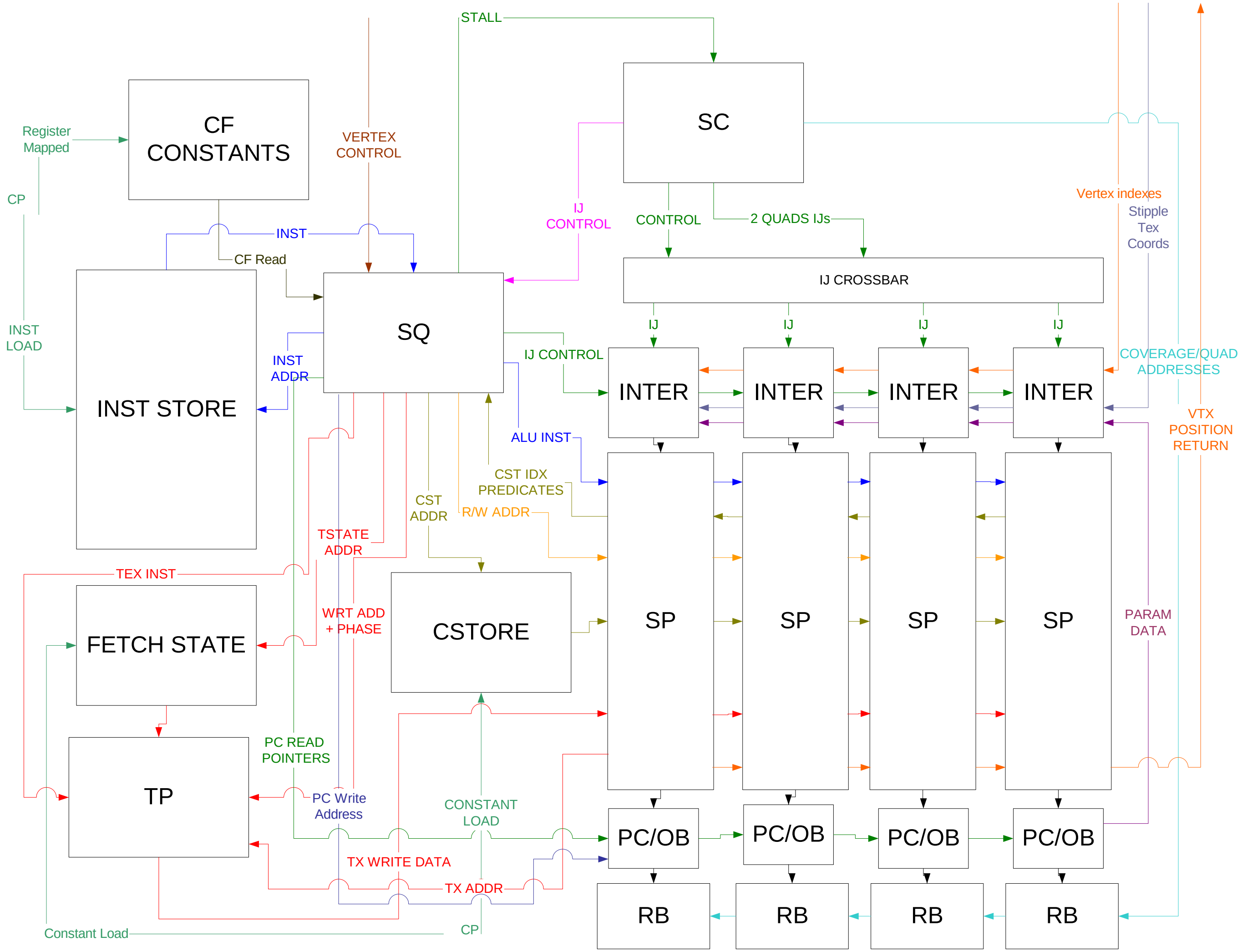


Figure 1: General Sequencer overview

1.1 Top Level Block Diagram

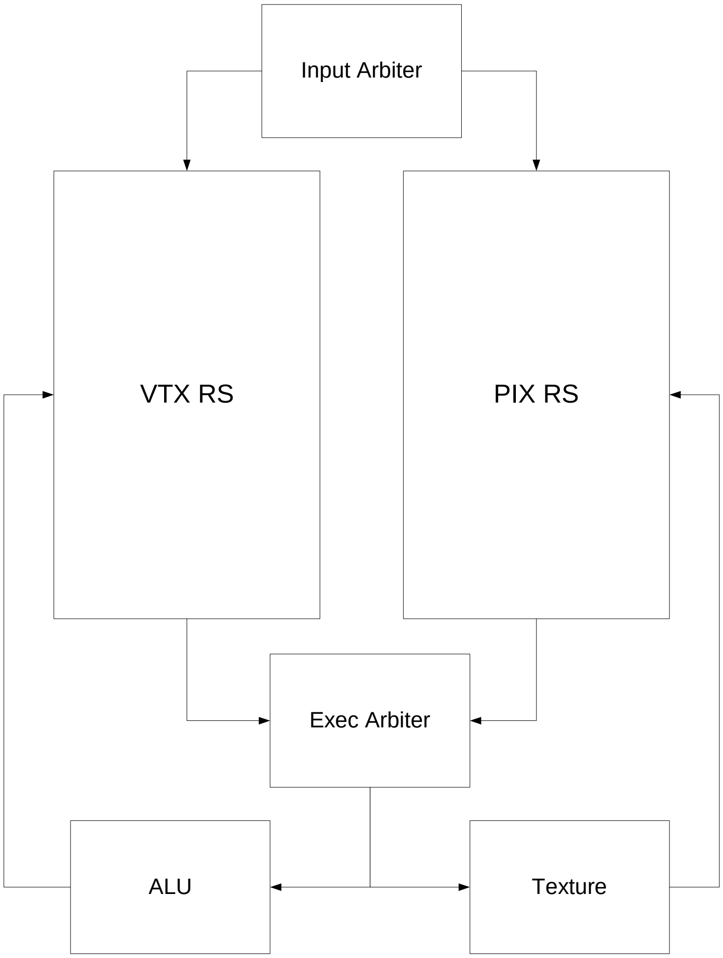


Figure 2: Reservation stations and arbiters

Under this new scheme, the sequencer (SQ) will only use one global state management machine per vector type (pixel, vertex) that we call the reservation station (RS).

1.2 Data Flow graph (SP)

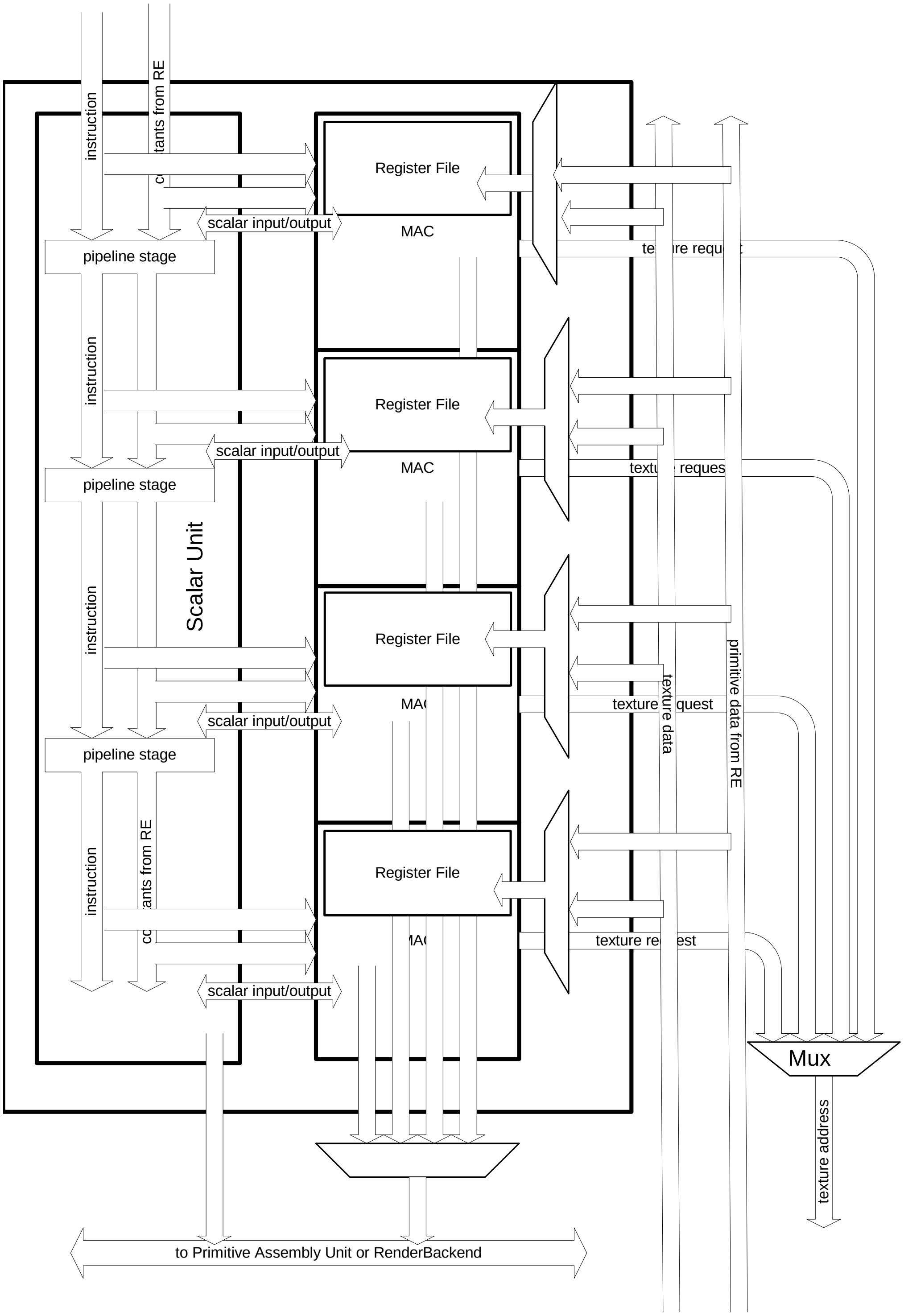


Figure 3: The shader Pipe

The gray area represents blocks that are replicated 4 times per shader pipe (16 times on the overall chip).

1.3 Control Graph

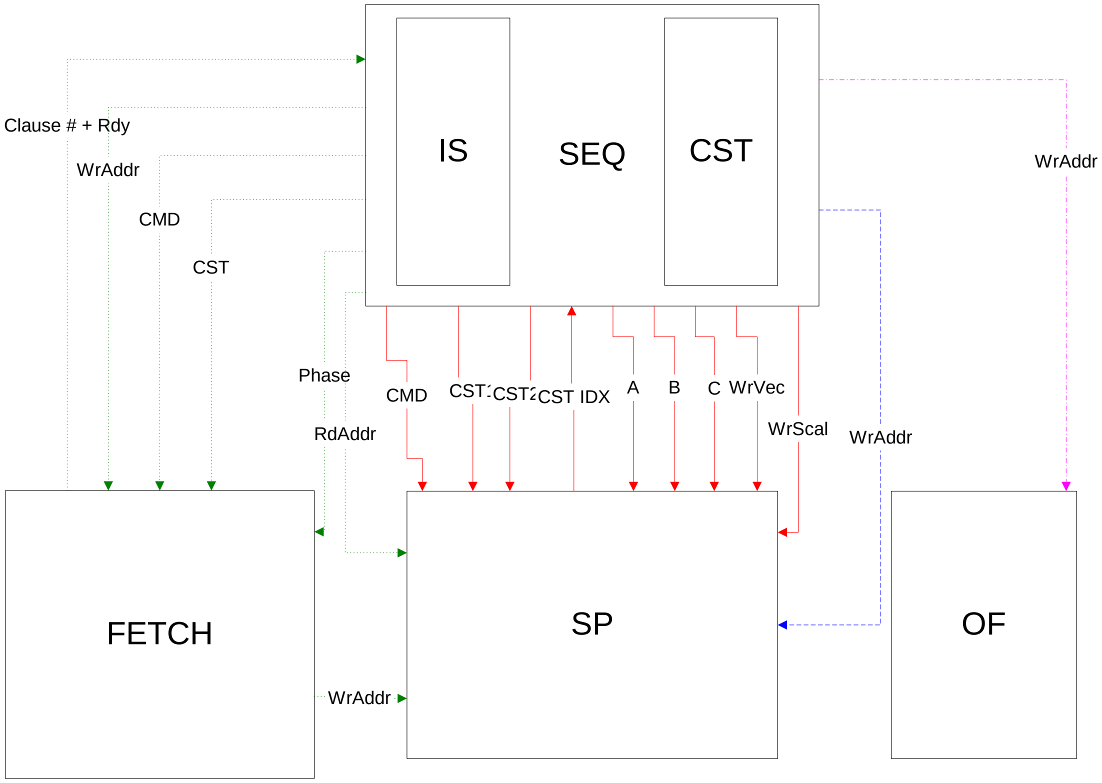


Figure 4: Sequencer Control interfaces

In green is represented the Fetch control interface, in red the ALU control interface, in blue the Interpolated/Vector control interface and in purple is the output file control interface.

2. Interpolated data bus

The interpolators contain an IJ buffer to pack the information as much as possible before writing it to the register file.

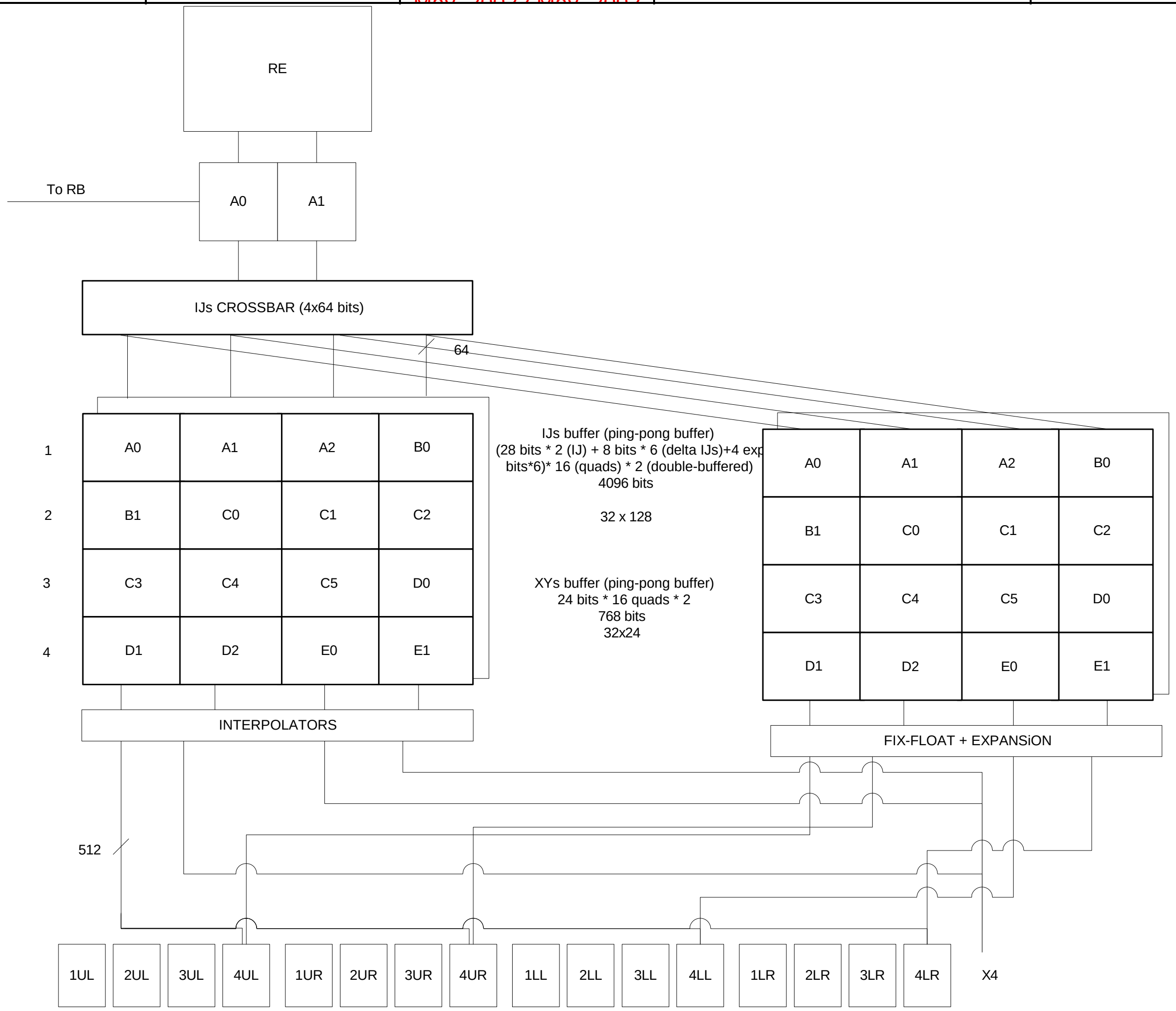


Figure 5: Interpolation buffers

WRITES

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21	T22	T23
SP 0	A0	A0	XY A0	B1	B1	XY B1	C3	C3	XY C3				D1	D1	XY D1									
SP 1	A1	A1	XY A1				C0	C0	XY C0	C4	C4	XY C4	D2	D2	XY D2									
SP 2	A2	A2	XY A2				C1	C1	XY C1	C5	C5	XY C5				E0	E0	XY E0						
SP 3				B0	B0	XY B0	C2	C2	XY C2				D0	D0	XY D0	E1	E1	XY E1						

READS

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21	T22	T23
SP 0	XY 0-3	XY 16-19	XY 32-35	XY 48-51	A0	B1	C3	D1					A0	B1	C3	D1					V 0-3	V 16-19	V 32-35	V 48-51
SP 1	XY 4-7	XY 20-23	XY 36-39	XY 52-55	A1		C4	D2		C0			A1		C4	D2		C0			V 4-7	V 20-23	V 36-39	V 52-55
SP 2	XY 8-11	XY 24-27	XY 40-43	XY 56-59	A2		C5			C1		E0	A2		C5			C1		E0	V 8-11	V 24-27	V 40-43	V 56-59
SP 3	XY 12-15	XY 28-31	XY 44-47	XY 60-63					B0	C2	D0	E1					B0	C2	D0	E1	V 12-15	V 28-31	V 44-47	V 60-63
		XY						P1									P2					VTX		

Figure 6: Interpolation timing diagram

Above is an example of a tile the sequencer might receive from the SC. The write side is how the data get stacked into the XY and IJ buffers, the read side is how the data is passed to the GPRs. The IJ information is packed in the IJ buffer 4 quads at a time or two clocks. The sequencer allows at any given time as many as four quads to interpolate a parameter. They all have to come from the same primitive. Then the sequencer controls the write mask to the GPRs to write the valid data in.

3. Instruction Store

There is going to be only one instruction store for the whole chip. It will contain 4096 instructions of 96 bits each.

It is likely to be a 1 port memory; we use 1 clock to load the ALU instruction, 1 clocks to load the Fetch instruction, 1 clock to load 2 control flow instructions and 1 clock to write instructions.

The instruction store is loaded by the CP thru the register mapped registers.

The VS_BASE and PS_BASE context registers are used to specify for each context where its shader is in the instruction memory.

For the Real time commands the story is quite the same but for some small differences. There are no wrap-around points for real time so the driver must be careful not to overwrite regular shader data. The shared code (shared subroutines) uses the same path as real time.

4. Sequencer Instructions

All control flow instructions and move instructions are handled by the sequencer only. The ALUs will perform NOPs during this time (MOV PV,PV, PS,PS) if they have nothing else to do.

5. Constant Stores

5.1 Memory organizations

A likely size for the ALU constant store is 1024x128 bits. The read BW from the ALU constant store is 128 bits/clock and the write bandwidth is 32 bits/clock (directed by the CP bus size not by memory ports).

The maximum logical size of the constant store for a given shader is 256 constants. Or 512 for the pixel/vertex shader pair. The size of the re-mapping table is 128 lines (each line addresses 4 constants). The write granularity is 4 constants or 512 bits. It takes 16 clocks to write the four constants. Real time requires 256 lines in the physical memory (this is physically register mapped).

The texture state is also kept in a similar memory. The size of this memory is 320x96 bits (128 texture states for regular mode, 32 states for RT). The memory thus holds 128 texture states (192 bits per state). The logical size exposes 32 different states total, which are going to be shared between the pixel and the vertex shader. The size of the re-mapping table to for the texture state memory is 32 lines (each line addresses 1 texture state lines in the real memory). The CP write granularity is 1 texture state lines (or 192 bits). The driver sends 512 bits but the CP ignores the top 320 bits. It thus takes 6 clocks to write the texture state. Real time requires 32 lines in the physical memory (this is physically register mapped).

The control flow constant memory doesn't sit behind a renaming table. It is register mapped and thus the driver must reload its content each time there is a change in the control flow constants. Its size is 320*32 because it must hold 8 copies of the 32 dwords of control flow constants and the loop construct constants must be aligned.

The constant re-mapping tables for texture state and ALU constants are logically register mapped for regular mode and physically register mapped for RT operation.

5.2 Management of the Control Flow Constants

The control flow constants are register mapped, thus the CP writes to the according register to set the constant, the SQ decodes the address and writes to the block pointed by its current base pointer (CF_WR_BASE). On the read side, one level of indirection is used. A register (SQ_CONTEXT_MISC.CF_RD_BASE) keeps the current base pointer to the control flow block. This register is copied whenever there is a state change. Should the CP write to CF after the state change, the base register is updated with the (current pointer number +1)% number of states. This way, if the CP doesn't write to CF the state is going to use the previous CF constants.

5.3 Management of the re-mapping tables

5.3.1 *R400 Constant management*

The sequencer is responsible to manage two re-mapping tables (one for the constant store and one for the texture state). On a state change (by the driver), the sequencer will broadside copy the contents of its re-mapping tables to a new one. We have 8 different re-mapping tables we can use concurrently.

The constant memory update will be incremental, the driver only need to update the constants that actually changed between the two state changes.

For this model to work in its simplest form, the requirement is that the physical memory MUST be at least twice as large as the logical address space + the space allocated for Real Time. In our case, since the logical address space is 512 and the reserved RT space can be up to 256 entries, the memory must be of sizes 1280 and above. Similarly the size of the texture store must be of 32*2+32 = 96 entries and above.

5.3.2 *Proposal for R400LE constant management*

To make this scheme work with only 512+256 = 768 entries, upon reception of a CONTROL packet of state + 1, the sequencer would check for SQ_IDLE and PA_IDLE and if both are idle will erase the content of state to replace it with the new state (this is depicted in Figure 8: De-allocation mechanism~~Figure 8: De-allocation mechanism~~~~Figure 8: De-allocation mechanism~~). Note that in the case a state is cleared a value of 0 is written to the corresponding de-allocation counter location so that when the SQ is going to report a state change, nothing will be de-allocated upon the first report.

The second path sets all context dirty bits that were used in the current state to 1 (thus allowing the new state to reuse these physical addresses if needed).

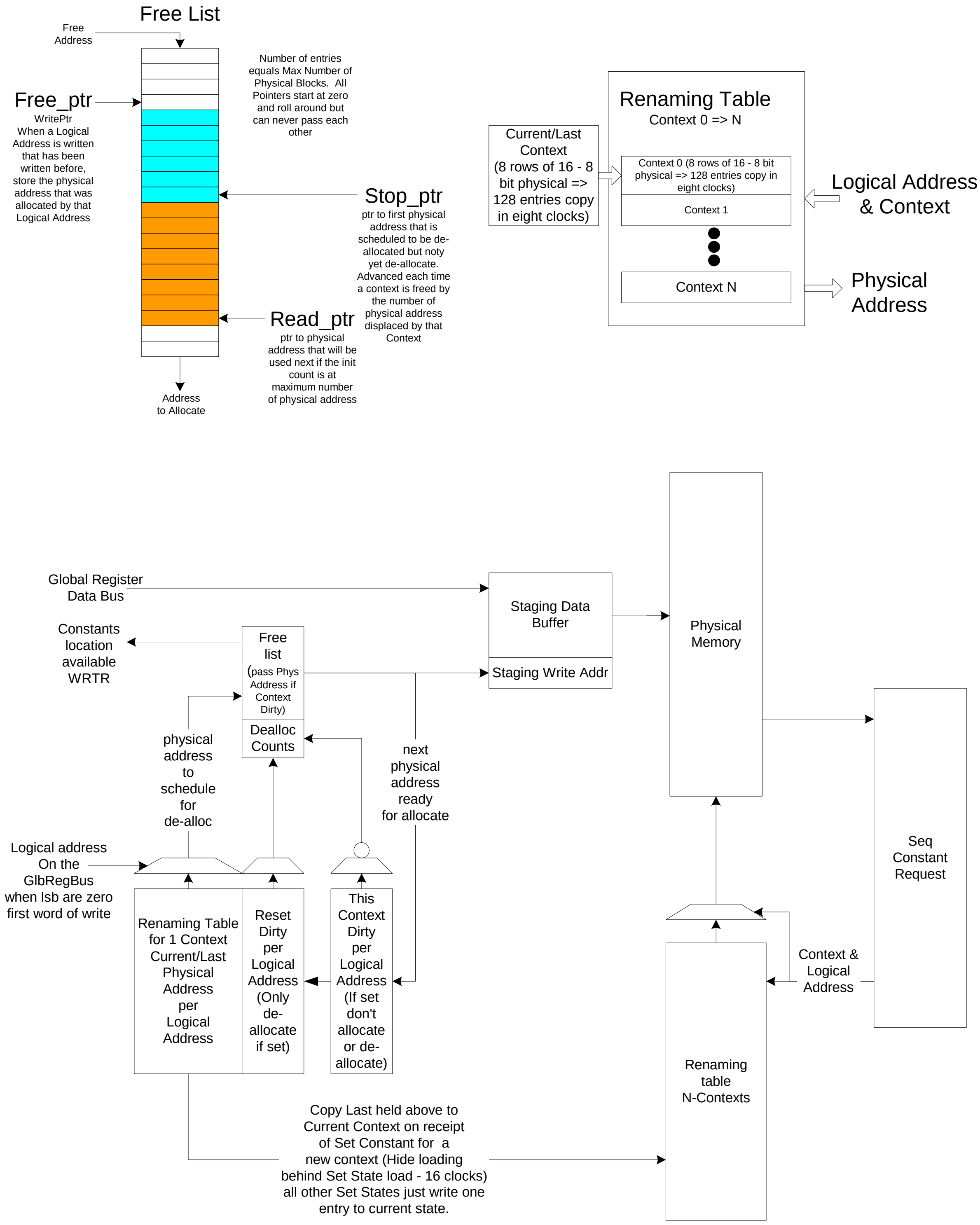

Free List
Free Address
Free_ptr
WritePtr
When a Logical Address is written that has been written before, store the physical address that was allocated by that Logical Address
Number of entries equals Max Number of Physical Blocks. All Pointers start at zero and roll around but can never pass each other
Stop_ptr
ptr to first physical address that is scheduled to be de-allocated but noty yet de-allocate. Advanced each time a context is freed by the number of physical address displaced by that Context
Read_ptr
ptr to physical address that will be used next if the init count is at maximum number of physical address
Address to Allocate
Current/Last Context (8 rows of 16 - 8 bit physical => 128 entries copy in eight clocks)
Renaming Table
Context 0 => N
Context 0 (8 rows of 16 - 8 bit physical => 128 entries copy in eight clocks)
Context 1
Context N
Logical Address & Context
Physical Address
Global Register Data Bus
Staging Data Buffer
Staging Write Addr
Physical Memory
Constants location available WRTR
Free list (pass Phys Address if Context Dirty)
Dealloc Counts
physical address to schedule for de-alloc
next physical address ready for allocate
Logical address On the GlbRegBus when lsb are zero first word of write
Renaming Table for 1 Context Current/Last Physical Address per Logical Address
Reset Dirty per Logical Address (Only de-allocate if set)
This Context Dirty per Logical Address (If set don't allocate or de-allocate)
Seq Constant Request
Context & Logical Address
Renaming table N-Contexts
Copy Last held above to Current Context on receipt of Set Constant for a new context (Hide loading behind Set State load - 16 clocks) all other Set States just write one entry to current state.

Figure 7: Constant management

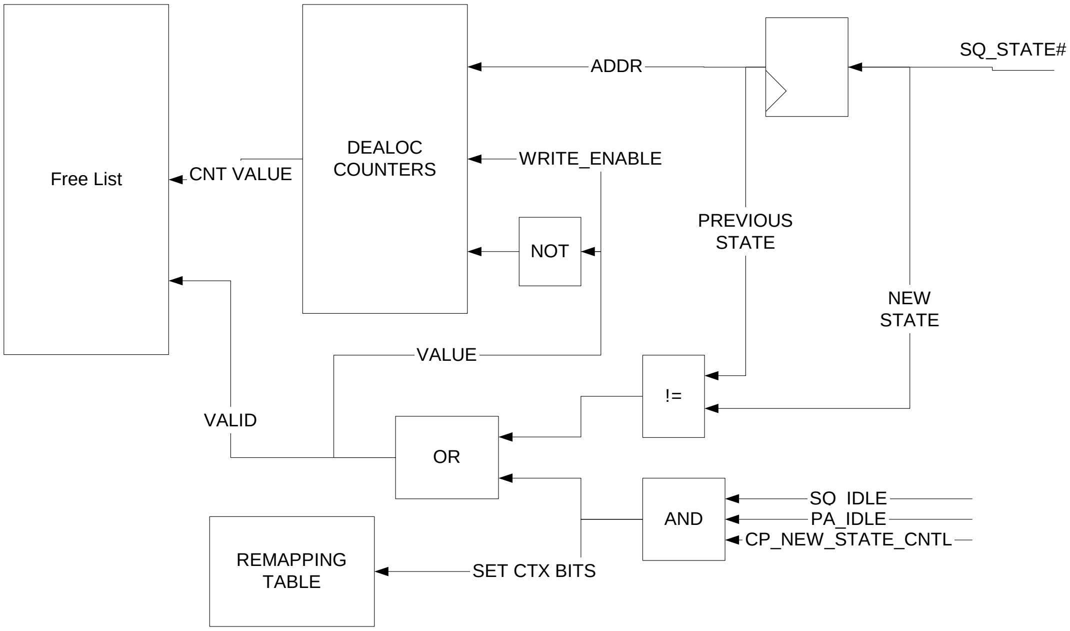


Figure 8: De-allocation mechanism for R400LE

5.3.3 *Dirty bits*

Two sets of dirty bits will be maintained per logical address. The first one will be set to zero on reset and set when the logical address is addressed. The second one will be set to zero whenever a new context is written and set for each address written while in this context. The reset dirty is not set, then writing to that logical address will not require de-allocation of whatever address stored in the renaming table. If it is set and the context dirty is not set, then the physical address store needs to be de-allocated and a new physical address is necessary to store the incoming data. If they are both set, then the data will be written into the physical address held in the renaming for the current logical address. No de-allocation or allocation takes place. This will happen when the driver does a set constant twice to the same logical address between context changes. NOTE: It is important to detect and prevent this, failure to do it will allow multiple writes to allocate all physical memory and thus hang because a context will not fit for rendering to start and thus free up space.

5.3.4 *Free List Block*

A free list block that would consist of a counter (called the IFC or Initial Free Counter) that would reset to zero and incremented every time a chunk of physical memory is used until they have all been used once. This counter would be checked each time a physical block is needed, and if the original ones have not been used up, us a new one, else check the free list for an available physical block address. The count is the physical address for when getting a chunk from the counter.

Storage of a free list big enough to store all physical block addresses.

Maintain three pointers for the free list that are reset to zero. The first one we will call write_ptr. This pointer will identify the next location to write the physical address of a block to be de-allocated. Note: we can never free more physical memory locations than we have. Once recording address the pointer will be incremented to walk the free list like a ring.

The second pointer will be called stop_ptr. The stop_ptr pointer will be advanced by the number of address chunks de-allocates when a context finishes. The address between the stop_ptr and write_ptr cannot be reused because they are still in use. But as soon as the context using then is dismissed the stop_ptr will be advanced.

The third pointer will be called read_ptr. This pointer will point will point to the next address that can be used for allocation as long as the read_ptr does not equal the stop_ptr and the IFC is at its maximum count.

5.3.5 *De-allocate Block*

This block will maintain a free physical address block count for each context. While in current context, a count shall be maintained specifying how many blocks were written into the free list at the write_ptr pointer. This count will be reset upon reset or when this context is active on the back and different than the previous context. It is actually a count of blocks in the previous context that will no longer be used. This count will be used to advance the write_ptr pointer to make available the set of physical blocks freed when the previous context was done. This allows the discard or de-allocation of any number of blocks in one clock.

5.3.6 *Operation of Incremental model*

The basic operation of the model would start with the write_ptr, stop_ptr, read_ptr pointers in the free list set to zero and the free list counter is set to zero. Also all the dirty bits and the previous context will be initialized to zero. When the first set constants happen, the reset dirty bit will not be set, so we will allocate a physical location from the free list counter because its not at the max value. The data will be written into physical address zero. Both the additional copy of the renaming table and the context zeros of the big renaming table will be updated for the logical address that was written by set start with physical address of 0. This process will be repeated for any logical address that are not dirty until the context changes. If a logical address is hit that has its dirty bits set while in the same context, both dirty bits would be set, so the new data will be over-written to the last physical address assigned for this logical address. When the first draw command of the context is detected, the previous context stored in the additional renaming table will be copied to the larger renaming table in the current (new) context location. Then the set constant logical address with be loaded with a new physical address during the copy and if the reset dirty was set, the physical address it replaced in the renaming table would be entered at the write_ptr pointer location on the free list and the write_ptr will be incremented. The de-allocation counter for the previous context (eight) will be incremented. This as set states come in for this context one of the following will happen:

1.) No dirty bits are set for the logical address being updated. A line will be allocated of the free-list counter or the free list at read_ptr pointer if read_ptr != to stop_ptr .
2.) Reset dirty set and Context dirty not set. A new physical address is allocated, the physical address in the renaming table is put on the free list at write_ptr and it is incremented along with the de-allocate counter for the last context.
3.) Context dirty is set then the data will be written into the physical address specified by the logical address.

This process will continue as long as set states arrive. This block will provide backpressure to the CP whenever he has not free list entries available (counter at max and stop_ptr == read_ptr). The command stream will keep a count of contexts of constants in use and prevent more than max constants contexts from being sent.

Whenever a draw packet arrives, the content of the re-mapping table is written to the correct re-mapping table for the context number. Also if the next context uses less constants than the current one all exceeding lines are moved to the free list to be de-allocated later. This happens in parallel with the writing of the re-mapping table to the correct memory.

Now preferable when the constant context leaves the last ALU clause it will be sent to this block and compared with the previous context that left. (Init to zero) If they differ than the older context will no longer be referenced and thus can be de-allocated in the physical memory. This is accomplished by adding the number of blocks freed this context to the stop_ptr pointer. This will make all the physical addresses used by this context available to the read_ptr allocate pointer for future allocation.

This device allows representation of multiple contexts of constants data with N copies of the logical address space. It also allows the second context to be represented as the first set plus some new additional data by just storing the delta's. It allows memory to be efficiently used and when the constants updates are small it can store multiple context. However, if the updates are large, less contexts will be stored and potentially performance will be degraded. Although it will still perform as well as a ring could in this case.

5.4 Constant Store Indexing

In order to do constant store indexing, the sequencer must be loaded first with the indexes (that come from the GPRs). There are 144 wires from the exit of the SP to the sequencer (9 bits pointers x 16 vertexes/clock). Since the data must pass thru the Shader pipe for the float to fixed conversion, there is a latency of 4 clocks (1 instruction)

between the time the sequencer is loaded and the time one can index into the constant store. The assembly will look like this

```
MOVA  R1.X,R2.X      // Loads the sequencer with the content of R2.X, also copies the content of R2.X into R1.X
NOP                  // latency of the float to fixed conversion
ADD   R3,R4,C0[R2.X]// Uses the state from the sequencer to add R4 to C0[R2.X] into R3
```

Note that we don't really care about what is in the brackets because we use the state from the MOVA instruction. R2.X is just written again for the sake of simplicity and coherency.

The storage needed in the sequencer in order to support this feature is 2*64*9 bits = 1152 bits.

5.5 Real Time Commands

The real time commands constants are written by the CP using the register mapped registers allocated for RT. It works is the same way than when dealing with regular constant loads BUT in this case the CP is not sending a logical address but rather a physical address and the reads are not passing thru the re-mapping table but are directly read from the memory. The boundary between the two zones is defined by the CONST_EO_RT control register. Similarly, for the fetch state, the boundary between the two zones is defined by the TSTATE_EO_RT control register.

5.6 Constant Waterfalling

In order to have a reasonable performance in the case of constant store indexing using the address register, we are going to have the possibility of using the physical memory port for read only. This way we can read 1 constant per clock and thus have a worst-case waterfall mode of 1 vertex per clock. There is a small synchronization issue related with this as we need for the SQ to make sure that the constants where actually written to memory (not only sent to the sequencer) before it can allow the first vector of pixels or vertices of the state to go thru the ALUs. To do so, the sequencer keeps 8 bits (one per render state) and sets the bits whenever the last render state is written to memory and clears the bit whenever a state is freed.

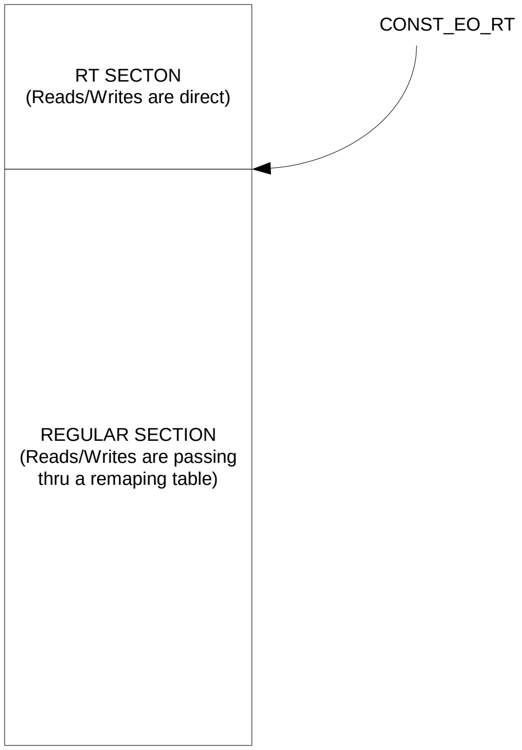


Figure 9: The instruction store

6. Looping and Branches

Loops and branches are planned to be supported and will have to be dealt with at the sequencer level. We plan on supporting constant loops and branches using a control program.

6.1 The controlling state.

The R400 controling state consists of:

Boolean[256:0]
Loop_count[7:0][31:0]
Loop_Start[7:0][31:0]
Loop_Step[7:0][31:0]

That is 256 Booleans and 32 loops.

We have a stack of 4 elements for nested calls of subroutines and 4 loop counters to allow for nested loops.

This state is available on a per shader program basis.

6.2 The Control Flow Program

We'd like to be able to code up a program of the form:

```
1:      Loop
2:      Exec    TexFetch
3:              TexFetch
4:              ALU
5:              ALU
6:              TexFetch
7:      End Loop
8:      ALU Export
```

But realize that 3: may be dependent on 2: and 4: is almost certainly dependent on 2: and 3:. Without clausing, these dependencies need to be expressed in the Control Flow instructions. Additionally, without separate 'texture clauses' and 'ALU clauses' we need to know which instructions to dispatch to the Texture Unit and which to the ALU unit. This information will be encapsulated in the flow control instructions.

Each control flow instruction will contain 2 bits of information for each (non-control flow) instruction:

a) ALU or Texture
b) Serialize Execution

(b) would force the thread to stop execution at this point (before the instruction is executed) and wait until all textures have been fetched. Given the allocation of reserved bits, this would mean that the count of an 'Exec' instruction would be limited to about 8 (non-control-flow) instructions. If more than this were needed, a second Exec (with the same conditions) would be issued.

Another function that relies upon 'clauses' is allocation and order of execution. We need to assure that pixels and vertices are exported in the correct order (even if not all execution is ordered) and that space in the output buffers are allocated in order. Additionally data can't be exported until space is allocated. A new control flow instruction:

Alloc <buffer select -- position,parameter, pixel or vertex memory. And the size required>.

would be created to mark where such allocation needs to be done. To assure allocation is done in order, the actual allocation for a given thread can not be performed unless the equivalent allocation for all previous threads is already completed. The implementation would also assure that execution of instruction(s) following the serialization due to the Alloc will occur in order -- at least until the next serialization or change from ALU to Texture. In most cases this will allow the exports to occur without any further synchronization. Only 'final' allocations or position allocations are

guaranteed to be ordered. Because strict ordering is required for pixels, parameters and positions, this implies only a single alloc for these structures. Vertex exports to memory do not require ordering during allocation and so multiple 'allocs' may be done.

6.2.1 *Control flow instructions table*

Here is the revised control flow instruction set.

Note that whenever a field is marked as RESERVED, it is assumed that all the bits of the field are cleared (0).

NOP		
47 … 44	43	42 … 0
0000	Addressing	RESERVED

This is a regular NOP.

Execute					
47 … 44~~47~~	43~~46… 43~~	40 … 34	33 …16	15…12	11 … 0
0001~~Addressing~~	Addressing~~0001~~	RESERVED	Instructions type + serialize (9 instructions)	Count	Exec Address

Execute_End					
47 … 44	43	40 … 34	33 …16	15…12	11 … 0
0010	Addressing	RESERVED	Instructions type + serialize (9 instructions)	Count	Exec Address

Execute up to 9 instructions at the specified address in the instruction memory. The Instruction type field tells the sequencer the type of the instruction (LSB) (1 = Texture, 0 = ALU and whether to serialize or not the execution (MSB) (1 = Serialize, 0 = Non-Serialized). If Execute_End this is the last execution block of the shader program.

~~This is a regular NOP.~~

Conditional_Execute						
47 … 44~~47~~	43~~46 … 43~~	42	41 … 34	33…16	15 …12	11 … 0
0011~~Addressing~~	Addressing~~0011~~	Condition	Boolean address	Instructions type + serialize (9 instructions)	Count	Exec Address

Conditional_Execute_End						
47 … 44	43	42	41 … 34	33…16	15 …12	11 … 0
0100	Addressing	Condition	Boolean address	Instructions type + serialize (9 instructions)	Count	Exec Address

If the specified Boolean (8 bits can address 256 Booleans) meets the specified condition then execute the specified instructions (up to 9 instructions). If the condition is not met, we go on to the next control flow instruction. If Conditional_Execute_End and the condition is met, this is the last execution block of the shader program.

Conditional_Execute_Predicates							
47 … 44~~47~~	43~~46 … 43~~	42	41 … 36	35 … 34	33…16	15…12	11 … 0
0101~~Addressing~~	Addressing~~0010~~	Condition	RESERVED	Predicate vector	Instructions type + serialize (9 instructions)	Count	Exec Address

Conditional_Execute_Predicates_End							
47 … 44	43	42	41 … 36	35 … 34	33…16	15…12	11 … 0
0110	Addressing	Condition	RESERVED	Predicate vector	Instructions type + serialize (9 instructions)	Count	Exec Address

Check the AND/OR of all current predicate bits. If AND/OR matches the condition execute the specified number of instructions. We need to AND/OR this with the kill mask in order not to consider the pixels that aren't valid. If the condition is not met, we go on to the next control flow instruction. If Conditional_Execute_Predicates_End and the condition is met, this is the last execution block of the shader program.

Loop_Start					
47 … 44~~47~~	43~~46 … 43~~	42 … 17	20 … 16	15…12	11 … 0
0111~~Addressing~~	Addressing~~0101~~	RESERVED	loop ID	RESERVED	Jump address

Loop Start. Compares the loop iterator with the end value. If loop condition not met jump to the address. Forward jump only. Also computes the index value. The loop id must match between the start to end, and also indicates which control flow constants should be used with the loop.

Loop_End						
47 … 44~~47~~	43~~46 … 43~~	42 … 24	23… 21	20 … 16	15…12	11 … 0
1000~~Addressing~~	Addressing~~0011~~	RESERVED	Predicate break	loop ID	RESERVED	start address

Loop end. Increments the counter by one, compares the loop count with the end value. If loop condition met, continue, else, jump BACK to the start of the loop. If predicate break != 0, then compares predicate vector n (specified by predicate break number). If all bits cleared then break the loop.

The way this is described does not prevent nested loops, and the inclusion of the loop id make this easy to do.

Conditionnal_Call						
47 … 44~~47~~	43~~46 … 43~~	42	41 … 34	33 … 13	12	11 … 0
1001~~Addressing~~	Addressing~~0111~~	Condition	Boolean address	RESERVED	Force Call	Jump address

If the condition is met, jumps to the specified address and pushes the control flow program counter on the stack. If force call is set the condition is ignored and the call is made always.

Return		
47 … 44~~47~~	43~~46 … 43~~	42 … 0
1010~~Addressing~~	Addressing~~1000~~	RESERVED

Pops the topmost address from the stack and jumps to that address. If nothing is on the stack, the program will just continue to the next instruction.

Conditionnal_Jump							
47 … 44~~47~~	43~~46 … 43~~	42	41… 34	33	32 … 13	12	11 … 0
1011~~Addressing~~	Addressing~~1001~~	Condition	Boolean address	FW only	RESERVED	Force Jump	Jump address

If force jump is set the condition is ignored and the jump is made always. If FW only is set then only forward jumps are allowed.

Allocate				
47 … 44~~47~~	43~~46 … 43~~	42…41	40 … 4	3 …0
1100~~Debug~~	Debug~~10 10~~	Buffer Select	RESERVED	Allocation size

Buffer Select takes a value of the following:
01 – position export (ordered export)
10 – parameter cache or pixel export (ordered export)
11 – pass thru (out of order exports).

If debug is set this is a debug alloc (ignore if debug DB_ON register is set to off).

~~Marks the end of the program.~~

6.3 Implementation

The envisioned implementation has a buffer that maintains the state of each thread. A thread lives in a given location in the buffer during its entire life, but the buffer has FIFO qualities in that threads leave in the order that they enter. Actually two buffers are maintained -- one for Vertices and one for Pixels. The intended implementation would allow for:

16 entries for vertices
48 entries for pixels.

From each buffer, arbitration logic attempts to select 1 thread for the texture unit and 1 (interleaved) thread for the ALU unit. Once a thread is selected it is read out of the buffer, marked as invalid, and submitted to appropriate execution unit. It is returned to the buffer (at the same place) with its status updated once all possible sequential instructions have been executed. A switch from ALU to TEX or visa-versa or a Serialize_Execution modifier forces the thread to be returned to the buffer.

Each entry in the buffer will be stored across two physical pieces of memory - most bits will be stored in a 1 read port device. Only bits needed for thread arbitration will be stored in a highly multi-ported structure. The bits kept in the 1 read port device will be termed 'state'. The bits kept in the multi-read ported device will be termed 'status'.

'State Bits' needed include:

1. Control Flow Instruction Pointer (13 bits),
2. Execution Count Marker 4 bits),
3. Loop Iterators (4x9 bits),
4. Call return pointers (4x12 bits),
5. Predicate Bits (64 bits),
6. Export ID (1 bit),
7. Parameter Cache base Ptr (7 bits),
8. GPR Base Ptr (8 bits),
9. Context Ptr (3 bits).
10. LOD corrections (6x16 bits)
11. Valid bits (64 bits)

Absent from this list are 'Index' pointers. These are costly enough that I'm presuming that they are instead stored in the GPRs. The first seven fields above (Control Flow Ptr, Execution Count, Loop Counts, call return ptrs, Predicate bits, PC base ptr and export ID) are updated every time the thread is returned to the buffer based on how much progress has been mode on thread execution. GPR Base Ptr, Context Ptr and LOD corrections are unchanged throughout execution of the thread.

'Status Bits' needed include:

- Valid Thread
- Texture/ALU engine needed
- Texture Reads are outstanding
- Waiting on Texture Read to Complete
- Allocation Wait (2 bits)
- 00 – No allocation needed
- 01 – Position export allocation needed (ordered export)
- 10 – Parameter or pixel export needed (ordered export)
- 11 – pass thru (out of order export)
- Allocation Size (4 bits)
- Position Allocated
- First thread of a new context
- Event thread (NULL thread that needs to trickle down the pipe)
- Last (1 bit)
- Pulse SX (1 bit)

All of the above fields from all of the entries go into the arbitration circuitry. The arbitration circuitry will select a winner for both the Texture Engine and for the ALU engine. There are actually two sets of arbitration -- one for pixels and one for vertices. A final selection is then done between the two. But the rest of this implementation summary only considers the 'first' level selection which is similar for both pixels and vertices.

Texture arbitration requires no allocation or ordering so it is purely based on selecting the 'oldest' thread that requires the Texture Engine.

ALU arbitration is a little more complicated. First, only threads where either of Texture_Reads_outstanding or Waiting_on_Texture_Read_to_Complete are '0' are considered. Then if Allocation_Wait is active, these threads are further filtered based on whether space is available. If the allocation is position allocation, then the thread is only considered if all 'older' threads have already done their position allocation (position allocated bits set). If the allocation is parameter or pixel allocation, then the thread is only considered if it is the oldest thread. Also a thread is not considered if it is a parameter or pixel or position allocation, has its First_thread_of_a_new_context bit set and would cause ALU interleaving with another thread performing the same parameter or pixel or position allocation. Finally the 'oldest' of the threads that pass through the above filters is selected. If the thread needed to allocate, then at this time the allocation is done, based on Allocation_Size. If a thread has its “last” bit set, then it is also removed from the buffer, never to return.

If I now redefine 'clauses' to mean 'how many times the thread is removed from the thread buffer for the purpose of exection by either the ALU or Texture engine', then the minimum number of clauses needed is 2 -- one to perform the allocation for exports (execution automatically halts after an 'Alloc' instruction) (but doesn't performs the actual allocation) and one for the actual ALU/export instructions. As the 'Alloc' instruction could be part of a texture clause (presumably the final instruction in such a clause), a thread could still execute in this minimal number of 2 clauses, even if it involved texture fetching.

The Texture_Reads_Outstanding bit must be updated by the sequencer, based on keeping track of how many Texture Clauses have been executed by a given thread that have not yet had there data returned. Any number above 0 results in this bit being set. We could consider forcing synchronization such that two texture clauses for a given thread may not be outstanding at any time (that would be my preference for simplicity reasons and because it would require only very little change in the texture pipe interface). This would allow the sequencer to set the bit on execution of the texture clause, and allow the texture unit to return a pointer to the thread buffer on completion that clears the bit.

6.4 Data dependant predicate instructions

Data dependant conditionals will be supported in the R400. The only way we plan to support those is by supporting three vector/scalar predicate operations of the form:

PRED_SETE_# - similar to SETE except that the result is 'exported' to the sequencer.
PRED_SETNE_# - similar to SETNE except that the result is 'exported' to the sequencer.
PRED_SETGT_# - similar to SETGT except that the result is 'exported' to the sequencer
PRED_SETGTE_# - similar to SETGTE except that the result is 'exported' to the sequencer

For the scalar operations only we will also support the two following instructions:
PRED_SETE0_# – SETE0
PRED_SETE1_# – SETE1

The export is a single bit - 1 or 0 that is sent using the same data path as the MOVA instruction. The sequencer will maintain 4 sets of 64 bit predicate vectors (in fact 8 sets because we interleave two programs but only 4 will be exposed) and use it to control the write masking. This predicate is not maintained across clause boundaries. The # sign is used to specify which predicate set you want to use 0 thru 3.

Then we have two conditional execute bits. The first bit is a conditional execute “on” bit and the second bit tells us if we execute on 1 or 0. For example, the instruction:

P0_ADD_# R0,R1,R2

Is only going to write the result of the ADD into those GPRs whose predicate bit is 0. Alternatively, P1_ADD_# would only write the results to the GPRs whose predicate bit is set. The use of the P0 or P1 without precharging the sequencer with a PRED instruction is undefined.

{Issue: do we have to have a NOP between PRED and the first instruction that uses a predicate?}

6.5 HW Detection of PV,PS

Because of the control program, the compiler cannot detect statically dependant instructions. In the case of non-masked writes and subsequent reads the sequencer will insert uses of PV,PS as needed. This will be done by comparing the read address and the write address of consecutive instructions. For masked writes, the sequencer will insert NOPs wherever there is a dependant read/write.

The sequencer will also have to insert NOPs between PRED_SET and MOVA instructions and their uses.

6.6 Register file indexing

Because we can have loops in fetch clause, we need to be able to index into the register file in order to retrieve the data created in a fetch clause loop and use it into an ALU clause. The instruction will include the base address for register indexing and the instruction will contain these controls:

Bit7	Bit 6	
0	0	'absolute register'
0	1	'relative register'
1	0	'previous vector'
1	1	'previous scalar'

In the case of an absolute register we just take the address as is. In the case of a relative register read we take the base address and we add to it the loop_index and this becomes our new address that we give to the shader pipe.

The sequencer is going to keep a loop index computed as such:

Index = Loop_iterator*Loop_step + Loop_start.

We loop until loop_iterator = loop_count. Loop_step is a signed value [-128…127]. The computed index value is a 10 bit counter that is also signed. Its real range is [-256,256]. The tenth bit is only there so that we can provide an out of range value to the “indexing logic” so that it knows when the provided index is out of range and thus can make the necessary arrangements.

6.7 Debugging the Shaders

In order to be able to debug the pixel/vertex shaders efficiently, we provide 2 methods.

6.7.1 *Method 1: Debugging registers*

Current plans are to expose 2 debugging, or error notification, registers:
1. address register where the first error occurred
2. count of the number of errors

The sequencer will detect the following groups of errors:
- count overflow
- constant indexing overflow
- register indexing overflow

Compiler recognizable errors:
  - jump errors
    relative jump address > size of the control flow program
  - call stack
    call with stack full
    return with stack empty

A jump error will always cause the program to break. In this case, a break means that a clause will halt execution, but allowing further clauses to be executed.

With all the other errors, program can continue to run, potentially to worst-case limits. The program will only break if the DB_PROB_BREAK register is set.

If indexing outside of the constant or the register range, causing an overflow error, the hardware is specified to return the value with an index of 0. This could be exploited to generate error tokens, by reserving and initializing the 0th register (or constant) for errors.

{ISSUE : Interrupt to the driver or not?}

6.7.2 *Method 2: Exporting the values in the GPRs*

1) The sequencer will have a debug active, count register and an address register for this mode.

Under the normal mode execution follows the normal course.

Under the debug mode it is assumed that the program is always exporting n debug vectors and that all other exports to the SX block (position, color, z, ect) will been turned off (changed into NOPs) by the sequencer (even if they occur before the address stated by the ADDR debug register).

7. Pixel Kill Mask

A vector of 64 bits is kept by the sequencer per group of pixels/vertices. Its purpose is to optimize the texture fetch requests and allow the shader pipe to kill pixels using the following instructions:

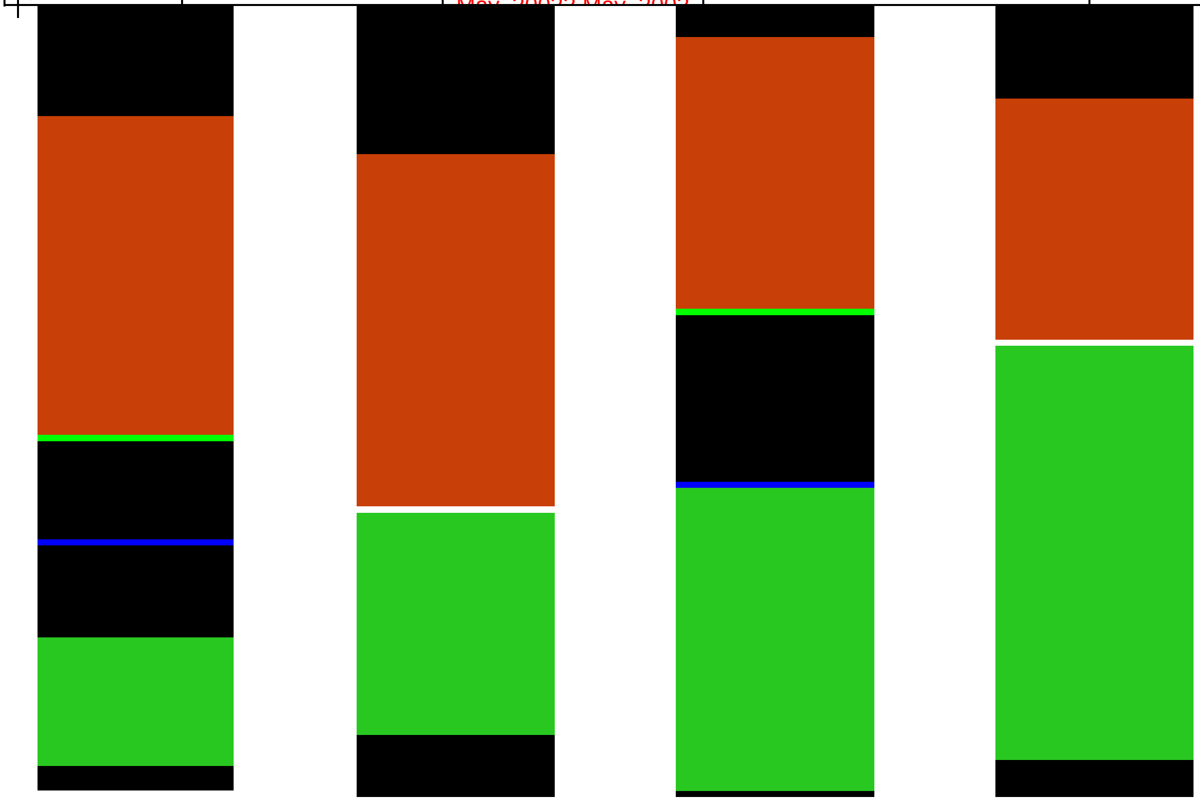
MASK_SETE
MASK_SETNE
MASK_SETGT
MASK_SETGTE

8. Multipass vertex shaders (HOS)

Multipass vertex shaders are able to export from the 6 last clauses but to memory ONLY.

9. Register file allocation

The register file allocation for vertices and pixels can either be static or dynamic. In both cases, the register file in managed using two round robins (one for pixels and one for vertices). In the dynamic case the boundary between pixels and vertices is allowed to move, in the static case it is fixed to 128-VERTEX_REG_SIZE for vertices and PIXEL_REG_SIZE for pixels.



Above is an example of how the algorithm works. Vertices come in from top to bottom; pixels come in from bottom to top. Vertices are in orange and pixels in green. The blue line is the tail of the vertices and the green line is the tail of the pixels. Thus anything between the two lines is shared. When pixels meets vertices the line turns white and the boundary is static until both vertices and pixels share the same “unallocated bubble”. Then the boundary is allowed to move again. The numbering of the GPRs starts from the bottom of the picture at index 0 and goes up to the top at index 127.

10. Fetch Arbitration

The fetch arbitration logic chooses one of the 8 potentially pending fetch clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute. Once chosen, the clause state machine will send one 2x2 fetch per clock (or 4 fetches in one clock every 4 clocks) until all the fetch instructions of the clause are sent. This means that there cannot be any dependencies between two fetches of the same clause.

The arbitrator will not wait for the fetches to return prior to selecting another clause for execution. The fetch pipe will be able to handle up to X(?) in flight fetches and thus there can be a fair number of active clauses waiting for their fetch return data.

11. ALU Arbitration

ALU arbitration proceeds in almost the same way than fetch arbitration. The ALU arbitration logic chooses one of the 8 potentially pending ALU clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute. There are two ALU arbiters, one for the even clocks and one for the odd clocks. For example, here is the sequencing of two interleaved ALU clauses (E and O stands for Even and Odd sets of 4 clocks):

Einst0 Oinst0 Einst1 Oinst1 Einst2 Oinst2 Einst0 Oinst3 Einst1 Oinst4 Einst2 Oinst0…
Proceeding this way hides the latency of 8 clocks of the ALUs. Also note that the interleaving also occurs across clause boundaries.

12. Handling Stalls

When the output file is full, the sequencer prevents the ALU arbitration logic from selecting the last clause (this way nothing can exit the shader pipe until there is place in the output file. If the packet is a vertex packet and the position buffer is full (POS_FULL) then the sequencer also prevents a thread from entering the exporting clause (3?). The sequencer will set the OUT_FILE_FULL signal n clocks before the output file is actually full and thus the ALU arbiter will be able read this signal and act accordingly by not preventing exporting clauses to proceed.

13. Content of the reservation station FIFOs

The reservation FIFOs contain the state of the vector of pixels and vertices. We have two sets of those: one for pixels, and one for vertices. They contain 3 bits of Render State 7 bits for the base address of the GPRs, some bits for LOD correction and coverage mask information in order to fetch fetch for only valid pixels, the quad address.

14. The Output File

The output file is where pixels are put before they go to the RBs. The write BW to this store is 256 bits/clock. Just before this output file are staging registers with write BW 512 bits/clock and read BW 256 bits/clock. The staging registers are 4x128 (and there are 16 of those on the whole chip).

15. IJ Format

The IJ information sent by the PA is of this format on a per quad basis:

We have a vector of IJ's (one IJ per pixel at the centroid of the fragment or at the center of the pixel depending on the mode bit). The interpolation is done at a different precision across the 2x2. The upper left pixel's parameters are always interpolated at full 20x24 mantissa precision. Then the result of the interpolation along with the difference in IJ in reduced precision is used to interpolate the parameter for the other three pixels of the 2x2. Here is how we do it:

Assuming P0 is the interpolated parameter at Pixel 0 having the barycentric coordinates I(0), J(0) and so on for P1,P2 and P3. Also assuming that A is the parameter value at V0 (interpolated with I), B is the parameter value at V1 (interpolated with J) and C is the parameter value at V2 (interpolated with (1-I-J).

$$\Delta 01I = I(1) - I(0)$$
$$\Delta 01J = J(1) - J(0)$$
$$\Delta 02I = I(2) - I(0)$$
$$\Delta 02J = J(2) - J(0)$$
$$\Delta 03I = I(3) - I(0)$$
$$\Delta 03J = J(3) - J(0)$$

P0	P1
P2	P3

$$P0 = C + I(0) * (A - C) + J(0) * (B - C)$$
$$P1 = P0 + \Delta 01I * (A - C) + \Delta 01J * (B - C)$$
$$P2 = P0 + \Delta 02I * (A - C) + \Delta 02J * (B - C)$$
$$P3 = P0 + \Delta 03I * (A - C) + \Delta 03J * (B - C)$$

P0 is computed at 20x24 mantissa precision and P1 to P3 are computed at 8X24 mantissa precision. So far no visual degradation of the image was seen using this scheme.

Multiplies (Full Precision): 2
Multiplies (Reduced precision): 6
Subtracts 19x24 (Parameters): 2

Adds: 8

FORMAT OF P0's IJ : Mantissa 20 Exp 4 for I + Sign
Mantissa 20 Exp 4 for J + Sign

FORMAT of Deltas (x3):Mantissa 8 Exp 4 for I + Sign
Mantissa 8 Exp 4 for J + Sign

Total number of bits : 20*2 + 8*6 + 4*8 + 4*2 = 128

All numbers are kept using the un-normalized floating point convention: if exponent is different than 0 the number is normalized if not, then the number is un-normalized. The maximum range for the IJs (Full precision) is +/- 63 and the range for the Deltas is +/- 127.

15.1 Interpolation of constant attributes

Because of the floating point imprecision, we need to take special provisions if all the interpolated terms are the same or if two of the barycentric coordinates are the same.

We start with the premise that if A = B and B = C and C = A, then P0,1,2,3 = A. Since one or more of the IJ terms may be zero, so we extend this to:

```
if (A=B and B=C and C=A)
   P0,1,2,3 = A;
else if ((I = 0) or (J = 0)) and
          ((J = 0) or (1-I-J = 0)) and
          ((1-J-I = 0) or (I = 0))) {
                if(I != 0) {
                    P0 = A;
               } else if(J != 0) {
                    P0 = B;
               } else {
                    P0 = C;
              }
             //rest of the quad interpolated normally
}
else
{
             normal interpolation
}
```

16. Staging Registers

In order for the reuse of the vertices to be 14, the sequencer will have to re-order the data sent IN ORDER by the VGT for it to be aligned with the parameter cache memory arrangement. Given the following group of vertices sent by the VGT:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 || 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 || 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 || 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63

The sequencer will re-arrange them in this fashion:

0 1 2 3 16 17 18 19 32 33 34 35 48 49 50 51 || 4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55 || 8 9 10 11 24 25 26 27 40 41 42 43 56 57 58 59 || 12 13 14 15 28 29 30 31 44 45 46 47 60 61 62 63

The || markers show the SP divisions. In the event a shader pipe is broken, the VGT will send padding to account for the missing pipe. For example, if SP1 is broken, vertices 4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55 will still be sent by the VGT to the SQ **BUT** will not be processed by the SP and thus should be considered invalid (by the SU and VGT).

The most straightforward, *non-compressed* interface method would be to convert, in the VGT, the data to 32-bit floating point prior to transmission to the VSISRs. In this scenario, the data would be transmitted to (and stored in) the VSISRs in full 32-bit floating point. This method requires three 24-bit fixed-to-float converters in the VGT. Unfortunately, it also requires and additional 3,072 bits of storage across the VSISRs. This interface is illustrated in Figure 11~~Figure 11Figure 11~~. The area of the fixed-to-float converters and the VSISRs for this method is roughly estimated as 0.759sqmm using the R300 process. The gate count estimate is shown in Figure 10~~Figure 10Figure 10~~.

Basis for 8-deep Latch Memory (from R300)			
8x24-bit	11631 μ^2		60.57813 μ^2 per bit
Area of 96x8-deep Latch Memory	46524 μ^2		
Area of 24-bit Fix-to-float Converter	4712 μ^2 per converter		
Method 1	Block	Quantity	Area
	F2F	3	14136
	8x96 Latch	16	744384
			758520 μ^2

Figure 10:Area Estimate for VGT to Shader Interface

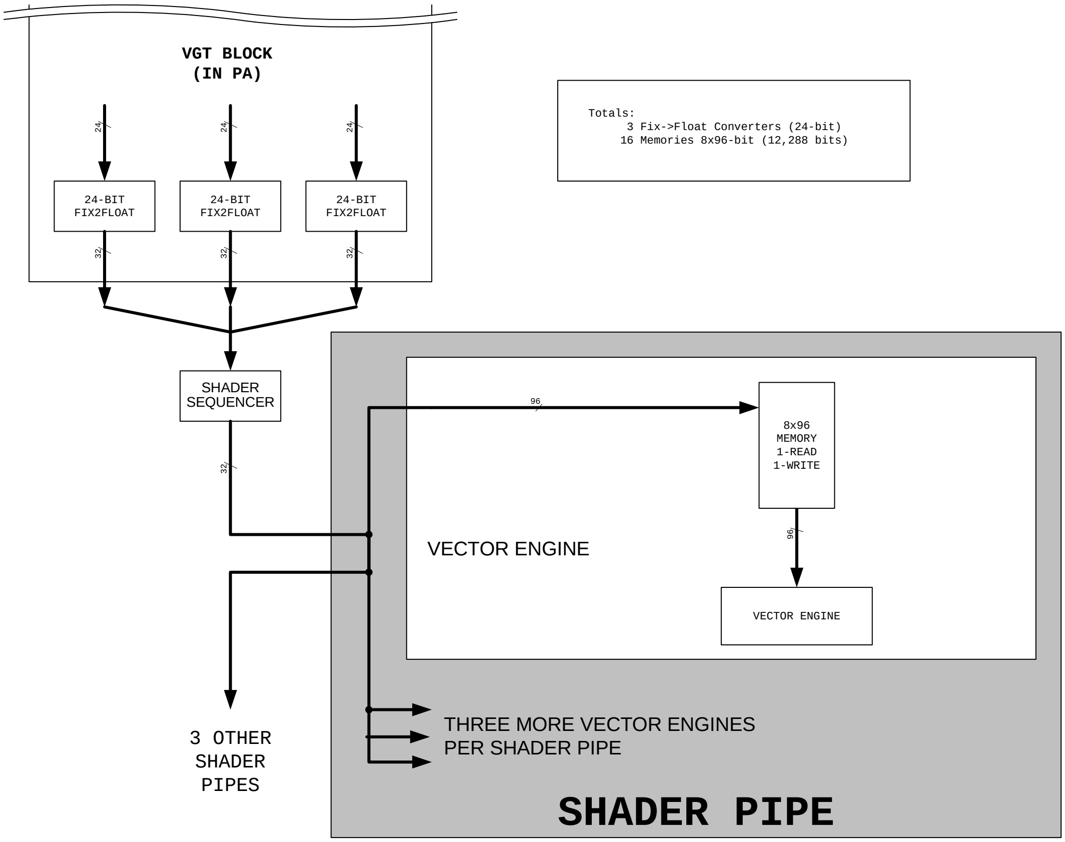


Figure 11:VGT to Shader Interface

17. The parameter cache

The parameter cache is where the vertex shaders export their data. It consists of 16 128x128 memories (1R/1W). The reuse engine will make it so that all vertexes of a given primitive will hit different memories. The allocation method for these memories is a simple round robin. The parameter cache pointers are mapped in the following way: 4MSBs are the memory number and the 7 LSBs are the address within this memory.

MEMORY NUMBER 4 bits	ADDRESS 7 bits

The PA generates the parameter cache addresses as the positions come from the SQ. All it needs to do is keep a Current_Location pointer (7 bits only) and as the positions comes increment the memory number. When the memory number field wraps around, the PA increments the Current_Location by VS_EXPORT_COUNT (a snooped register from the SQ). As an example, say the memories are all empty to begin with and the vertex shader is exporting 8 parameters per vertex (VS_EXPORT_COUNT = 8). The first position received is going to have the PC address 00000000000 the second one 00010000000, third one 00100000000 and so on up to 11110000000. Then the next position received (the 17th) is going to have the address 00000001000, the 18th 00010001000, the 19th 00100001000 and so on. The Current_location is NEVER reset BUT on chip resets. The only thing to be careful about is that if the SX doesn't send you a full group of positions (<64) then you need to fill the address space so that the next group starts correctly aligned (for example if you receive only 33 positions then you need to add 2*VS_EXPORT_COUNT to Current_Location and reset the memory count to 0 before the next vector begins).

17.1 Export restrictions

17.1.1 *Pixel exports:*

Pixels can export 1,2,3 or 4 color buffers to the SX(+z). The exports will be done in order. The PRED_OPTIMIZE function has to be turned of if the exports are done using interleaved predicated instructions. The exports will always be ordered to the SX.

17.1.2 *Vertex exports:*

Position or parameter caches can be exported in any order in the shader program. It is always better to export posistion as soon as possible. Position has to be exported in a single export block (no texture instructions can be placed between the exports). Parameter cache exports can be done in any order with texture instructions interleaved. The PRED_OPTIMIZE function has to be turned of if the exports are done using interleaved predicated instructions to the Parameter cache (see Arbitration restrictions for details). The exports will always be allocated in order to the SX.

17.1.3 *Pass thru exports:*

Pass thru exports have to be done in groups of the form:

```
Alloc 4 (8 or 12)
Execute ALU(ADDR) ALU(DATA) ALU(DATA) ALU(DATA)…
```

They cannot have texture instructions interleaved in the export block. These exports **are not guaranteed to be ordered**.

Also, when doing a pass thru export, Position MUST be exported AFTER all pass thru exports. This position export is used to synchronize the chip when doing a transition from pass thru shader to regular shader and vice versa.

17.2 Arbitration restrictions

Here are the Sequencer arbitration restrictions:

1) Cannot execute a serialized thread if the corresponding texture pending bit is set
2) Cannot allocate position if any older thread has not allocated position
3) If last thread is marked as not valid AND marked as last and we are about to execute the second to oldest thread also marked last then:
    a. Both threads must be from the same context (cannot allow a first thread)
    b. Must turn off the predicate optimization for the second thread
4) Cannot execute a texture clause if texture reads are pending
5) Cannot execute last if texture pending (even if not serial)

18. Export Types

The export type (or the location where the data should be put) is specified using the destination address field in the ALU instruction. Here is a list of all possible export modes:

18.1 Vertex Shading

0:15	- 16 parameter cache
16:31	- Empty (Reserved?)
32	- Export Address
33:40	- 8 vertex exports to the frame buffer and index
41:47	- Empty
48:55	- 8 debug export (interpret as normal vertex export)
60	- export addressing mode
61	- Empty
62	- position

63	- sprite size export that goes with position export (point_h,point_w,edgeflag,misc)

18.2 Pixel Shading

0	- Color for buffer 0 (primary)
1	- Color for buffer 1
2	- Color for buffer 2
3	- Color for buffer 3
4:7	- Empty
8	- Buffer 0 Color/Fog (primary)
9	- Buffer 1 Color/Fog
10	- Buffer 2 Color/Fog
11	- Buffer 3 Color/Fog
12:15	- Empty
16:31	- Empty (Reserved?)
32	- Export Address
33:40	- 8 exports for multipass pixel shaders.
41:47	- Empty
48:55	- 8 debug exports (interpret as normal pixel export)
60	- export addressing mode
61:62	- Empty
63	- Z for primary buffer (Z exported to 'alpha' component)

19. Special Interpolation modes

19.1 Real time commands

We are unable to use the parameter memory since there is no way for a command stream to write into it. Instead we need to add three 16x128 memories (one for each of three vertices x 16 interpolants). These will be mapped onto the register bus and written by type 0 packets, and output to the the parameter busses (the sequencer and/or PA need to be able to address the realtime parameter memory as well as the regular parameter store. For higher performance we should be able able to view them as two banks of 16 and do double buffering allowing one to be loaded, while the other is rasterized with. Most overlay shaders will need 2 or 4 scalar coordinates, one option might be to restrict the memory to 16x64 or 32x64 allowing only two interpolated scalars per cycle, the only problem I see with this is, if we view support for 16 vector-4 interpolants important (true only if we map Microsoft's high priority stream to the realtime stream), then the PA/sequencer need to support a realtime-specific mode where we need to address 32 vectors of parameters instead of 16. This mode is triggered by the primitive type: REAL TIME. The actual memories are in the in the SX blocks. The parameter data memories are hooked on the RBBM bus and are loaded by the CP using register mapped memory.

19.2 Sprites/ XY screen coordinates/ FB information

When working with sprites, one may want to overwrite the parameter 0 with SC generated data. Also, XY screen coordinates may be needed in the shader program. This functionality is controlled by the gen_I0 register (in SQ) in conjunction with the SND_XY register (in SC). Also it is possible to send the faceness information (for OGL front/back special operations) to the shader using the same control register. Here is a list of all the modes and how they interact together:

Gen_st is a bit taken from the interface between the SC and the SQ. This is the MSB of the primitive type. If the bit is set, it means we are dealing with Point AA, Line AA or sprite and in this case the vertex values are going to generated between 0 and 1.

Param_Gen_I0 disable, snd_xy disable, no gen_st – I0 = No modification
Param_Gen_I0 disable, snd_xy disable, gen_st – I0 = No modification
Param_Gen_I0 disable, snd_xy enable, no gen_st – I0 = No modification
Param_Gen_I0 disable, snd_xy enable, gen_st – I0 = No modification
Param_Gen_I0 enable, snd_xy disable, no gen_st – I0 = garbage, garbage, garbage, faceness

Param_Gen_I0 enable, snd_xy disable, gen_st – I0 = garbage, garbage, s, t
Param_Gen_I0 enable, snd_xy enable, no gen_st – I0 = screen x, screen y, garbage, faceness
Param_Gen_I0 enable, snd_xy enable, gen_st – I0 = screen x, screen y, s, t

19.3 Auto generated counters

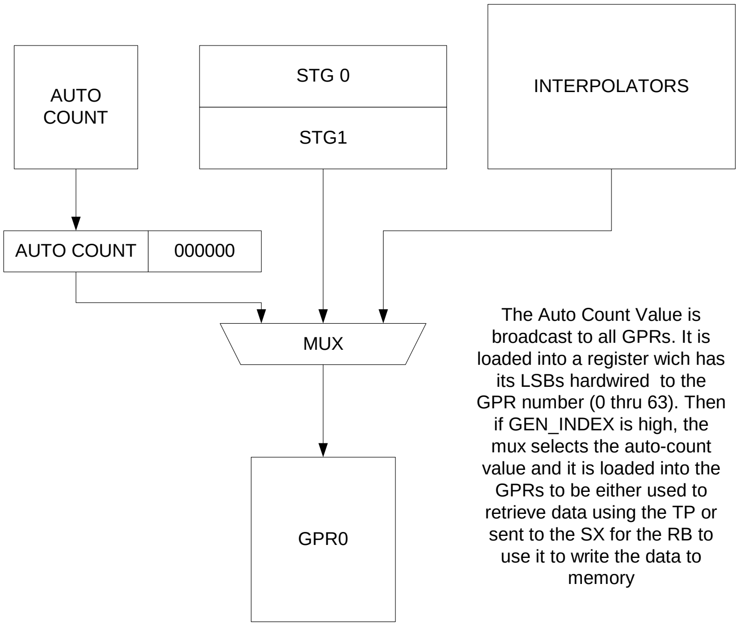
In the cases we are dealing with multipass shaders, the sequencer is going to generate a vector count to be able to both use this count to write the 1st pass data to memory and then use the count to retrieve the data on the 2nd pass. The count is always generated in the same way but it is passed to the shader in a slightly different way depending on the shader type (pixel or vertex). This is toggled on and off using the GEN_INDEX register. The sequencer is going to keep two counters, one for pixels and one for vertices. Every time a full vector of vertices or pixels is written to the GPRs the counter is incremented. Every time a state change is detected, the corresponding counter is reset. While there is only one count broadcast to the GPRs, the LSB are hardwired to specific values making the index different for all elements in the vector.

19.3.1 *Vertex shaders*

In the case of vertex shaders, if GEN_INDEX is set, the data will be put into the x field of the third register (it means that the compiler must allocate 3 GPRs in all multipass vertex shader modes).

19.3.2 *Pixel shaders*

In the case of pixel shaders, if GEN_INDEX is set and Param_Gen_I0 is enabled, the data will be put in the x field of the 2nd register (R1.x), else if GEN_INDEX is set the data will be put into the x field of the 1st register (R0.x).



The Auto Count Value is broadcast to all GPRs. It is loaded into a register wich has its LSBs hardwired to the GPR number (0 thru 63). Then if GEN_INDEX is high, the mux selects the auto-count value and it is loaded into the GPRs to be either used to retrieve data using the TP or sent to the SX for the RB to use it to write the data to memory

Figure 12: GPR input mux Control

20. State management

Every clock, the sequencer will report to the CP the oldest states still in the pipe. These are the states of the programs as they enter the last ALU clause.

20.1 Parameter cache synchronization

In order for the sequencer not to begin a group of pixels before the associated group of vertices has finished, the sequencer will keep a 6 bit count per state (for a total of 8 counters). These counters are initialized to 0 and every time a vertex shader exports its data TO THE PARAMETER CACHE, the corresponding pointer is incremented. When the SC sends a new vector of pixels with the SC_SQ_new_vector bit asserted, the sequencer will first check if the count is greater than 0 before accepting the transmission (it will in fact accept the transmission but then lower its ready to receive). Then the sequencer waits for the count to go to one and decrements it. The sequencer can then issue the group of pixels to the interpolators. Every time the state changes, the new state counter is initialized to 0.

21. XY Address imports

The SC will be able to send the XY addresses to the GPRs. It does so by interleaving the writes of the IJs (to the IJ buffer) with XY writes (to the XY buffer). Then when writing the data to the GPRs, the sequencer is going to interpolate the IJ data or pass the XY data thru a Fix→float converter and expander and write the converted values to the GPRs. The Xys are currently SCREEN SPACE COORDINATES. The values in the XY buffers will wrap. See section 19.2 for details on how to control the interpolation in this mode.

21.1 Vertex indexes imports

In order to import vertex indexes, we have 16 8x96 staging registers. These are loaded one line at a time by the VGT block (96 bits). They are loaded in floating point format and can be transferred in 4 or 8 clocks to the GPRs.

22. Registers

22.1 Control

REG_DYNAMIC	Dynamic allocation (pixel/vertex) of the register file on or off.
REG_SIZE_PIX	Size of the register file's pixel portion (minimal size when dynamic allocation turned on)
REG_SIZE_VTX	Size of the register file's vertex portion (minimal size when dynamic allocation turned on)
ARBITRATION_POLICY	policy of the arbitration between vertexes and pixels
INST_BASE_VTX	start point for the vertex instruction store (RT always ends at vertex_base and Begins at 0)
INST_BASE_PIX	start point for the pixel shader instruction store
ONE_THREAD	debug state register. Only allows one program at a time into the GPRs
ONE_ALU	debug state register. Only allows one ALU program at a time to be executed (instead of 2)
INSTRUCTION	This is where the CP puts the base address of the instruction writes and type (auto-incremented on reads/writes) Register mapped
CONSTANTS	512*4 ALU constants + 32*6 Texture state 32 bits registers (logically mapped)
CONSTANTS_RT	256*4 ALU constants + 32*6 texture states? (physically mapped)
CONSTANT_EO_RT	This is the size of the space reserved for real time in the constant store (from 0 to CONSTANT_EO_RT). The re-mapping table operates on the rest of the memory

TSTATE_EO_RT This is the size of the space reserved for real time in the fetch state store (from 0 to TSTATE_EO_RT). The re-mapping table operates on the rest of the memory

22.2 Context

PS_BASE	base pointer for the pixel shader in the instruction store
VS_BASE	base pointer for the vertex shader in the instruction store
VS_CF_SIZE	size of the vertex shader (# of instructions in control program/2)
PS_CF_SIZE	size of the pixel shader (# of instructions in control program/2)
PS_SIZE	size of the pixel shader (cntl+instructions)
VS_SIZE	size of the vertex shader (cntl+instructions)
PS_NUM_REG	number of GPRs to allocate for pixel shader programs
VS_NUM_REG	number of GPRs to allocate for vertex shader programs
PARAM_SHADE	One 16 bit register specifying which parameters are to be gouraud shaded (0 = flat, 1 = gouraud)
PARAM_WRAP	64 bits: for which parameters (and channels (xyzw)) do we do the cyl wrapping (0=linear, 1=cylindrical).
PS_EXPORT_MODE	0xxxx : Normal mode

	1xxxx : Multipass mode If normal, bbbz where bbb is how many colors (0-4) and z is export z or not If multipass 1-12 exports for color.
VS_EXPORT_MODE	0: position (1 vector), 1: position (2 vectors), 3:multipass
VS_EXPORT_COUNT	Number of locations exported by the VS (and thus number of interpolated parameters)
PARAM_GEN_I0	Do we overwrite or not the parameter 0 with XY data and generated T and S values
GEN_INDEX	Auto generates an address from 0 to XX. Puts the results into R0-1 for pixel shaders and R2 for vertex shaders
CONST_BASE_VTX (9 bits)	Logical Base address for the constants of the Vertex shader
CONST_BASE_PIX (9 bits)	Logical Base address for the constants of the Pixel shader
CONST_SIZE_PIX (8 bits)	Size of the logical constant store for pixel shaders
CONST_SIZE_VTX (8 bits)	Size of the logical constant store for vertex shaders
INST_PRED_OPTIMIZE	Turns on the predicate bit optimization (if of, conditional_execute_predicates is always executed).
CF_BOOLEANS	256 boolean bits
CF_LOOP_COUNT	32x8 bit counters (number of times we traverse the loop)
CF_LOOP_START	32x8 bit counters (init value used in index computation)
CF_LOOP_STEP	32x8 bit counters (step value used in index computation)

23. DEBUG Registers

23.1 Context

DB_PROB_ADDR	instruction address where the first problem occurred
DB_PROB_COUNT	number of problems encountered during the execution of the program
DB_PROB_BREAK	break the clause if an error is found.
DB_ON	turns on an off debug method 2
DB_INST_COUNT	instruction counter for debug method 2
DB_BREAK_ADDR	break address for method number 2

23.2 Control

DB_ALUCST_MEMSIZE	Size of the physical ALU constant memory
DB_TSTATE_MEMSIZE	Size of the physical texture state memory

24. Interfaces

24.1 External Interfaces

Whenever an x is used, it means that the bus is broadcast to all units of the same name. For example, if a bus is named SQ→SPx it means that SQ is going to broadcast the same information to all SP instances.

24.2 SC to SP Interfaces

24.2.1 *SC_SP#*

There is one of these interfaces at front of each of the SP (buffer to stage pixel interpolators). This interface transmits the I,J data for pixel interpolation. For the entire system, two quads per clock are transferred to the 4 SPs, so each of these 4 interfaces transmits one half of a quad per clock. The interface below describes a half of a quad worth of data.
The actual data which is transferred per quad is
Ref Pix I => S4.20 Floating Point I value
Ref Pix J => S4.20 Floating Point J value

Delta Pix I (x3) => S4.8 Floating Point Delta I value
Delta Pix J (x3) => S4.8 Floating Point Delta J value
This equates to a total of 128 bits which transferred over 2 clocks
and therefor needs an interface 64 bits wide

Additionally, X,Y data (12-bit unsigned fixed) is conditionally sent across this data bus over the same wires in an additional clock. The X,Y data is sent on the lower 24 bits of the data bus with faceness in the msb.
Transfers across these interfaces are synchronized with the SC_SQ IJ Control Bus transfers.

The data transfer across each of these busses is controlled by a IJ_BUF_INUSE_COUNT in the SC. Each time the SC has sent a pixel vector's worth of data to the SPs, he will increment the IJ_BUF_INUSE_COUNT count. Prior to sending the next pixel vectors data, he will check to make sure the count is less than MAX_BUFER_MINUS_2, if not the SC will stall until the SQ returns a pipelined pulse to decrement the count when he has scheduled a buffer free. Note: We could/may optimize for the case of only sending only IJ to use all the buffers to pre-load more. Currently it is planned for the SP to hold 2 double buffers of I,J data and two buffers of X,Y data, so if either X,Y or Centers and Centroids are on, then the SC can send two Buffers.

In at least the initial version, the SC shall send 16 quads per pixel vector even if the vector is not full. This will increment buffer write address pointers correctly all the time. (We may revisit this for both the SX,SP,SQ and add a EndOfVector signal on all interfaces to quit early. We opted for the simple mode first with a belief that only the end of packet and multiple new vector signals should cause a partial vector and that this would not really be significant performance hit.)

Name	**Bits**	**Description**
SC_SP#_data	64	IJ information sent over 2 clocks (or X,Y in 24 LSBs with faceness in upper bit) **Type 0 or 1**, First clock I, second clk J Field ULC URC LLC LRC Bits [63:39] [38:26] [25:13] [12:0] Format SE4M20 SE4M8 SE4M8 SE4M8 **Type 2** Field Face X Y Bits [63] [23:12] [11:0] Format Bit Unsigned Unsigned
SC_SP#_valid	1	Valid
SC_SP#_last_quad_data	1	This bit will be set on the last transfer of data per quad.
SC_SP#_type	2	0 -> Indicates centroids 1 -> Indicates centers 2 -> Indicates X,Y Data and faceness on data bus The SC shall look at state data to determine how many types to send for the interpolation process.

The # is included for clarity in the spec and will be replaced with a prefix of u#_ in the verilog module statement for the SC and the SP block will have neither because the instantiation will insert the prefix.

24.2.2 *SC_SQ*

This is the control information sent to the sequencer in order to synchronize and control the interpolation and/or loading data into the GPRs needed to execute a shader program on the sent pixels. This data will be sent over two clocks per transfer with 1 to 16 transfers. Therefore the bus (approx ~~92~~ 94 bits) could be folded in half to approx ~~47~~ 49 bits.

Name	**Bits**	**Description**
SC_SQ_data	46	Control Data sent to the SQ 1 clk transfers Event – valid data consist of event_id and state_id. Instruct SQ to post an event vector to send state id and event_id through request fifo

		and onto the reservation stations making sure state id and/or event_id gets back to the CP. Events only follow end of packets so no pixel vectors will be in progress. Empty Quad Mask – Transfer Control data consisting of pc_dealloc or new_vector. Receipt of this is to transfer pc_dealloc or new_vector without any valid quad data. New vector will always be posted to request fifo and pc_dealloc will be attached to any pixel vector outstanding or posted in request fifo if no valid quad outstanding. 2 clk transfers Quad Data Valid – Sending quad data with or without new_vector or pc_dealloc. New vector will be posted to request fifo with or without a pixel vector and pc_dealloc will be posted with a pixel vector unless none is in progress. In this case the pc_dealloc will be posted in the request queue. Filler quads will be transferred with The Quad mask set but the pixel corresponding pixel mask set to zero.
SC_SQ_valid	1	SC sending valid data, 2nd clk could be all zeroes

SC_SQ_data – first clock and second clock transfers are shown in the table below.

Name	**BitField**	**Bits**	**Description**
1st Clock Transfer			
SC_SQ_event	0	1	This transfer is a 1 clock event vector Force quad_mask = new_vector=pc_dealloc=0
SC_SQ_event_id	[4:1]	4	This field identifies the event 0 => denotes an End Of State Event 1 => TBD
SC_SQ_pc_dealloc	[7:5]	3	Deallocation token for the Parameter Cache
SC_SQ_new_vector	8	1	The SQ must wait for Vertex shader done count > 0 and after dispatching the Pixel Vector the SQ will decrement the count.
SC_SQ_quad_mask	[12:9]	4	Quad Write mask left to right SP0 => SP3
SC_SQ_end_of_prim	13	1	End Of the primitive
SC_SQ_state_id	[16:14]	3	State/constant pointer (6*3+3)
SC_SQ_pix_mask	[32:17]	16	Valid bits for all pixels SP0=>SP3 (UL,UR,LL,LR)
SC_SQ_provok_vtx	[37:36]	2	Provoking vertex for flat shading
SC_SQ_pc_ptr0	[48:38]	11	Parameter Cache pointer for vertex 0
2nd Clock Transfer			
SC_SQ_pc_ptr1	[10:0]	11	Parameter Cache pointer for vertex 1
SC_SQ_pc_ptr2	[21:11]	11	Parameter Cache pointer for vertex 2
SC_SQ_lod_correct	[45:22]	24	LOD correction per quad (6 bits per quad)
SC_SQ_prim_type	[48:46]	3	Stippled line and Real time command need to load tex cords from alternate buffer 000: Normal 100: Realtime 101: Line AA 110: Point AA (Sprite)

Name	Bits	Description
SQ_SC_free_buff	1	Pipelined bit that instructs SC to decrement count of buffers in use.
SQ_SC_dec_cntr_cnt	1	Pipelined bit that instructs SC to decrement count of new vector and/or event sent to prevent SC from overflowing SQ interpolator/Reservation request fifo.

The scan converter will submit a partial vector whenever:

1.) He gets a primitive marked with an end of packet signal.
2.) A current pixel vector is being assembled with at least one or more valid quads and the vector has been marked for deallocate when a primitive marked new_vector arrives. The Scan Converter will submit a partial vector (up to 16quads with zero pixel mask to fill out the vector) prior to submitting the new_vector marker\primitive.

(This will prevent a hang which can be demonstrated when all primitives in a packet three vectors are culled except for a one quad primitive that gets marked pc_dealloc (vertices maximum size). In this case two new_vectors are submitted and processed, but then one valid quad with the pc_dealloc creates a vector and then the new would wait for another vertex vector to be processed, but the one being waited for could never export until the pc_dealloc signal made it through and thus the hang.)

24.2.3 *SQ to SX: Interpolator bus*

Name	Direction	Bits	Description
SQ_SXx_interp_flat_vtx	SQ→SPx	2	Provoking vertex for flat shading
SQ_SXx_interp_flat_gouraud	SQ→SPx	1	Flat or gouraud shading
SQ_SXx_interp_cyl_wrap	SQ→SPx	4	Wich channel needs to be cylindrical wrapped
SQ_SXx_pc_ptr0	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_pc_ptr1	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_pc_ptr2	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_rt_sel	SQ→SXx	1	Selects between RT and Normal data
SQ_SXx_pc_wr_en	SQ→SXx	1	Write enable for the PC memories
SQ_SXx_pc_wr_addr	SQ→SXx	7	Write address for the PCs
SQ_SXx_pc_channel_mask	SQ→SXx	4	Channel mask

24.2.4 *SQ to SP: Staging Register Data*

This is a broadcast bus that sends the VSISR information to the staging registers of the shader pipes.

Name	Direction	Bits	Description
SQ_SPx_vsr_data	SQ→SPx	96	Pointers of indexes or HOS surface information
SQ_SPx_vsr_double	SQ→SPx	1	0: Normal 96 bits per vert 1: double 192 bits per vert
SQ_SP0_ vsr_ valid	SQ→SP0	1	Data is valid
SQ_SP1_ vsr_ valid	SQ→SP1	1	Data is valid
SQ_SP2_ vsr_ valid	SQ→SP2	1	Data is valid
SQ_SP3_ vsr_ valid	SQ→SP3	1	Data is valid
SQ_SPx_vsr_read	SQ→SPx	1	Increment the read pointers

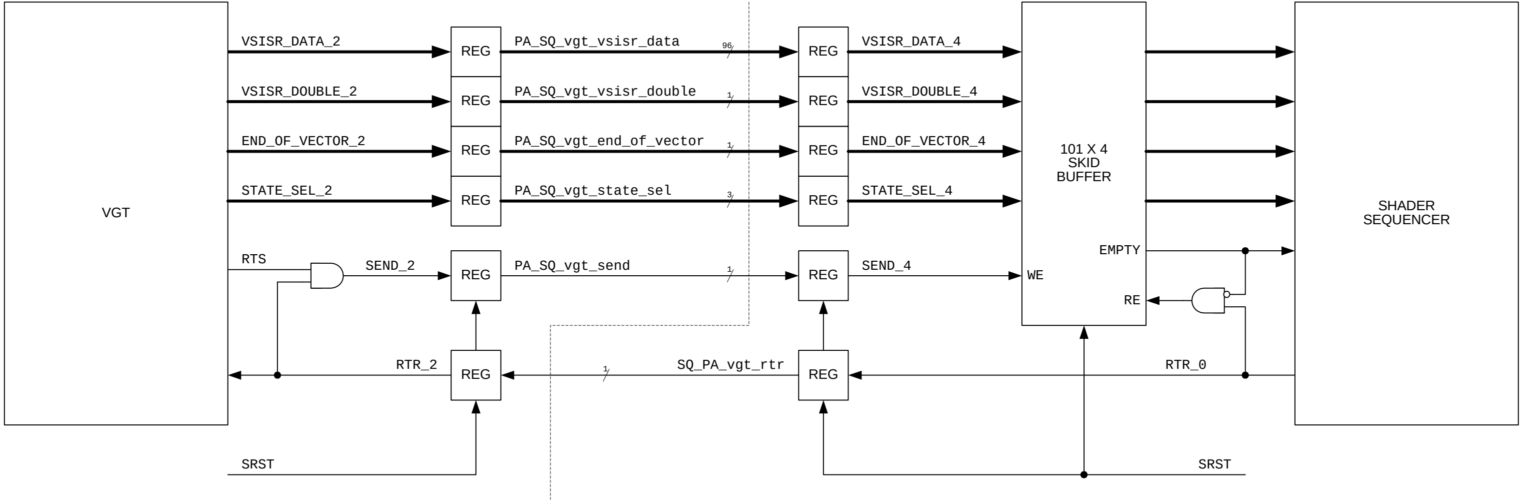
24.2.5 *VGT to SQ : Vertex interface*

24.2.5.1 Interface Signal Table

The area difference between the two methods is not sufficient to warrant complicating the interface or the state requirements of the VSISRs. **Therefore, the POR for this interface is that the VGT will transmit the data to the VSISRs (via the Shader Sequencer) in full, 32-bit floating-point format.** The VGT can transmit up to six 32-bit floating-point values to each VSISR where four or more values require two transmission clocks. The data bus is 96 bits wide.

Name	Bits	Description
VGT_SQ_vsisr_data	96	Pointers of indexes or HOS surface information
VGT_SQ_vsisr_double	1	0: Normal 96 bits per vert 1: double 192 bits per vert
VGT_SQ_end_of_vector	1	Indicates the last VSISR data set for the current process vector (for double vector data, "end_of_vector" is set on the first vector)
VGT_SQ_indx_valid	1	Vsisr data is valid
VGT_SQ_state	3	Render State (6*3+3 for constants). This signal is guaranteed to be correct when “VGT_SQ_vgt_end_of_vector” is high.
VGT_SQ_send	1	Data on the VGT_SQ is valid receive (see write-up for standard R400 SEND/RTR interface handshaking)
SQ_VGT_rtr	1	Ready to receive (see write-up for standard R400 SEND/RTR interface handshaking)

24.2.5.2 Interface Diagrams



VGT
VSISR_DATA_2
VSISR_DOUBLE_2
END_OF_VECTOR_2
STATE_SEL_2
RTS
SEND_2
RTR_2
SRST
REG
PA_SQ_vgt_vsisr_data
96
PA_SQ_vgt_vsisr_double
1
PA_SQ_vgt_end_of_vector
1
PA_SQ_vgt_state_sel
3
PA_SQ_vgt_send
1
SQ_PA_vgt_rtr
1
VSISR_DATA_4
VSISR_DOUBLE_4
END_OF_VECTOR_4
STATE_SEL_4
SEND_4
101 X 4
SKID
BUFFER
EMPTY
WE
RE
RTR_0
SRST
SHADER
SEQUENCER

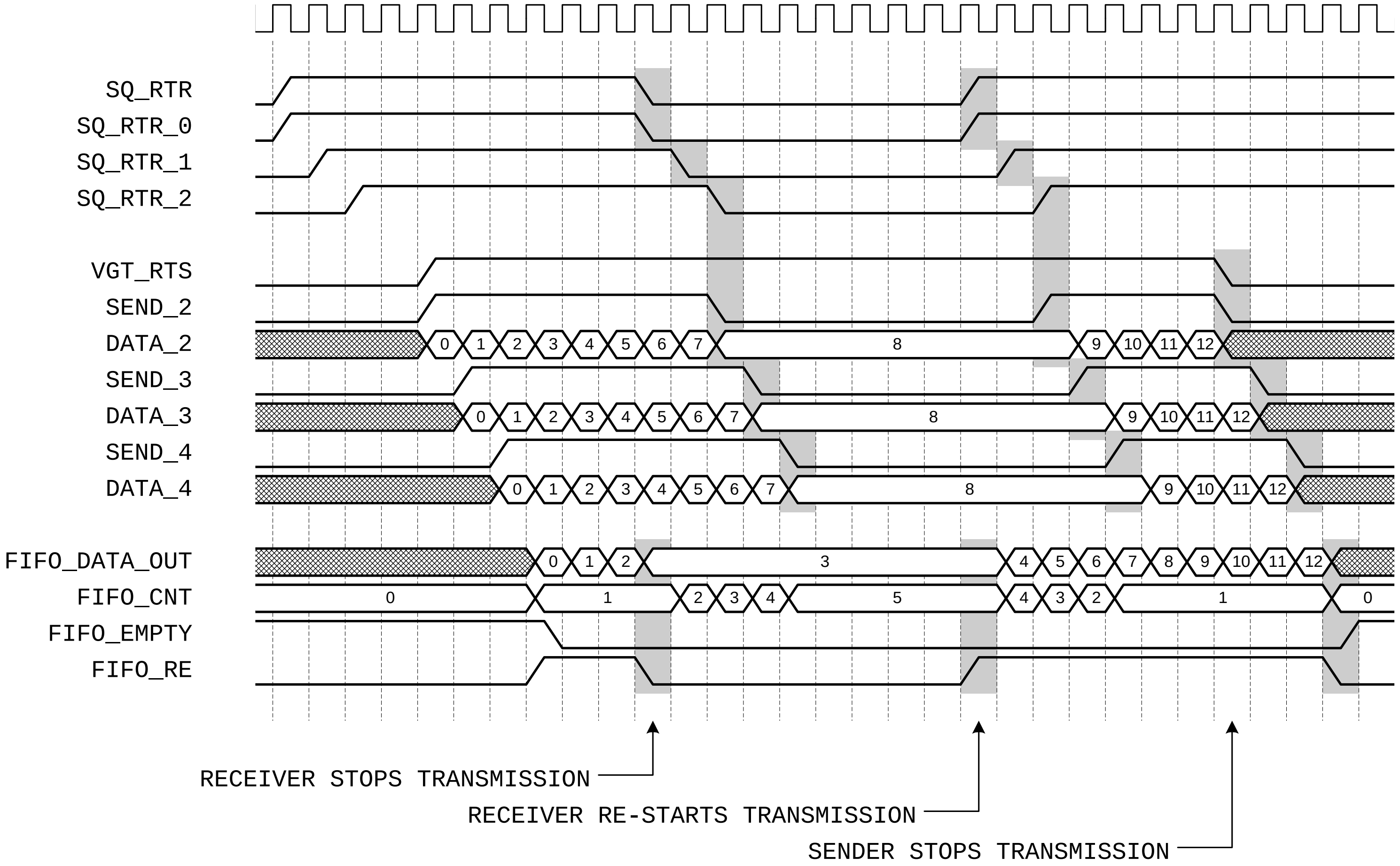


Figure 1. Detailed Logical Diagram for PA_SQ_vgt Interface.

24.2.6 *SQ to SX: Control bus*

Name	Direction	Bits	Description
SQ_SXx_exp_type	SQ→SXx	2	00: Pixel without z (1 to 4 buffers) 01: Pixel with z (1 to 4 buffers) 10: Position (1 or 2 results) 11: Pass thru (4,8 or 12 results aligned)
SQ_SXx_exp_number	SQ→SXx	2	Number of locations needed in the export buffer (encoding depends on the type see bellow).
SQ_SXx_exp_alu_id	SQ→SXx	1	ALU ID
SQ_SXx_exp_valid	SQ→SXx	1	Valid bit
SQ_SXx_exp_state	SQ→SXx	3	State Context
SQ_SXx_free_done	SQ→SXx	1	Pulse to indicate that the previous export is finished (this can be sent with or without the other fields of the interface)
SQ_SXx_free_alu_id	SQ→SXx	1	ALU ID

Depending on the type the number of export location changes:

- Type 00 : Pixels without Z
  - 00 = 1 buffer
  - 01 = 2 buffers
  - 10 = 3 buffers
  - 11 = 4 buffer
- Type 01: Pixels with Z
  - 00 = 2 Buffers (color + Z)
  - 01 = 3 buffers (2 color + Z)
  - 10 = 4 buffers (3 color + Z)
  - 11 = 5 buffers (4 color + Z)
- Type 10 : Position export
  - 00 = 1 position
  - 01 = 2 positions
  - 1X = Undefined
- Type 11: Pass Thru
  - 00 = 4 buffers
  - 01 = 8 buffers
  - 10 = 12 buffers
  - 11 = Undefined

Below the thick black line is the end of transfer packet that tells the SX that a given export is finished. The report packet **will always arrive either before or at the same time than the next export to the same ALU id**.

24.2.7 *SX to SQ : Output file control*

Name	Direction	Bits	Description
SXx_SQ_exp_count_rdy	SXx→SQ	1	Raised by SX0 to indicate that the following two fields reflect the result of the most recent export
SXx_SQ_exp_pos_avail	SXx→SQ	1	Specifies whether there is room for another position.
SXx_SQ_exp_buf_avail	SXx→SQ	7	Specifies the space available in the output buffers. 0: buffers are full 1: 2K-bits available (32-bits for each of the 64 pixels in a clause) ... 64: 128K-bits available (16 128-bit entries for each of 64 pixels) 65-127: RESERVED

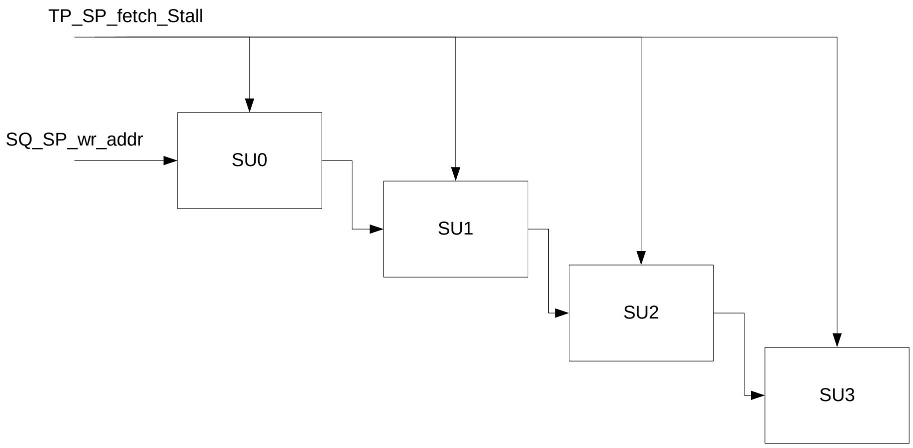
24.2.8 *SQ to TP: Control bus*

Once every clock, the fetch unit sends to the sequencer on which RS line it is now working and if the data in the GPRs is ready or not. This way the sequencer can update the fetch valid bits flags for the reservation station. The sequencer also provides the instruction and constants for the fetch to execute and the address in the register file where to write the fetch return data.

Name	Direction	Bits	Description
TPx_SQ_data_rdy	TPx→ SQ	1	Data ready
TPx_SQ_rs_line_num	TPx→ SQ	6	Line number in the Reservation station
TPx_SQ_type	TPx→ SQ	1	Type of data sent (0:PIXEL, 1:VERTEX)
SQ_TPx_send	SQ→TPx	1	Sending valid data
SQ_TPx_const	SQ→TPx	48	Fetch state sent over 4 clocks (192 bits total)
SQ_TPx_instr	SQ→TPx	24	Fetch instruction sent over 4 clocks
SQ_TPx_end_of_group	SQ→TPx	1	Last instruction of the group
SQ_TPx_Type	SQ→TPx	1	Type of data sent (0:PIXEL, 1:VERTEX)
SQ_TPx_gpr_phase	SQ→TPx	2	Write phase signal
SQ_TP0_lod_correct	SQ→TP0	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP0_pix_mask	SQ→TP0	4	Pixel mask 1 bit per pixel
SQ_TP1_lod_correct	SQ→TP1	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP1_pix_mask	SQ→TP1	4	Pixel mask 1 bit per pixel
SQ_TP2_lod_correct	SQ→TP2	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP2_pix_mask	SQ→TP2	4	Pixel mask 1 bit per pixel
SQ_TP3_lod_correct	SQ→TP3	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP3_pix_mask	SQ→TP3	4	Pixel mask 1 bit per pixel
SQ_TPx_rs_line_num	SQ→TPx	6	Line number in the Reservation station
SQ_TPx_write_gpr_index	SQ->TPx	7	Index into Register file for write of returned Fetch Data

24.2.9 *TP to SQ: Texture stall*

The TP sends this signal to the SQ and the SPs when its input buffer is full.



Name	Direction	Bits	Description
TP_SQ_fetch_stall	TP→ SQ	1	Do not send more texture request if asserted

24.2.10 SQ to SP: Texture stall

Name	Direction	Bits	Description
SQ_SPx_fetch_stall	SQ→SPx	1	Do not send more texture request if asserted

24.2.11 SQ to SP: GPR and auto counter

Name	Direction	Bits	Description
SQ_SPx_gpr_wr_addr	SQ→SPx	7	Write address
SQ_SPx_gpr_rd_addr	SQ→SPx	7	Read address
SQ_SPx_gpr_rd_en	SQ→SPx	1	Read Enable
SQ_SP0_gpr_wr_en	SQ→SPx	1	Write Enable for the GPRs of SP0
SQ_SP1_gpr_wr_en	SQ→SPx	1	Write Enable for the GPRs of SP1
SQ_SP2_gpr_wr_en	SQ→SPx	1	Write Enable for the GPRs of SP2
SQ_SP3_gpr_wr_en	SQ→SPx	1	Write Enable for the GPRs of SP3
SQ_SPx_gpr_phase	SQ→SPx	2	The phase mux (arbitrates between inputs, ALU SRC reads and writes)
SQ_SPx_channel_mask	SQ→SPx	4	The channel mask
SQ_SPx_gpr_input_sel	SQ→SPx	2	When the phase mux selects the inputs this tells from which source to read from: Interpolated data, VTX0, VTX1, autogen counter.
SQ_SPx_auto_count	SQ→SPx	12?	Auto count generated by the SQ, common for all shader pipes

24.2.12 *SQ to SPx: Instructions*

Name	Direction	Bits	Description
SQ_SPx_instr_start	SQ→SPx	1	Instruction start
SQ_SP_instr	SQ→SPx	21	Transferred over 4 cycles 0: SRC A Select 2:0 SRC A Argument Modifier 3:3 SRC A swizzle 11:4 VectorDst 17:12 Unused 20:18 --- - 1: SRC B Select 2:0 SRC B Argument Modifier 3:3 SRC B swizzle 11:4 ScalarDst 17:12 Unused 20:18 --- - 2: SRC C Select 2:0 SRC C Argument Modifier 3:3 SRC C swizzle 11:4 Unused 20:12 --- - 3: Vector Opcode 4:0 Scalar Opcode 10:5 Vector Clamp 11:11 Scalar Clamp 12:12 Vector Write Mask 16:13 Scalar Write Mask 20:17
SQ_SPx_exp_alu_id	SQ→SPx	1	ALU ID
SQ_SPx_exporting	SQ→SPx	2	0: Not Exporting 1: Vector Exporting 2: Scalar Exporting
SQ_SPx_stall	SQ→SPx	1	Stall signal
SQ_SP0_write_mask	SQ→SP0	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP1_ write_mask	SQ→SP1	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP2_ write_mask	SQ→SP2	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP3_ write_mask	SQ→SP3	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SPx_last	SQ→SPx	1	Last instruction of the block

24.2.13 *SP to SQ: Constant address load/ Predicate Set*

Name	Direction	Bits	Description
SP0_SQ_const_addr	SP0→SQ	36	Constant address load / predicate vector load (4 bits only) to the sequencer
SP0_SQ_valid	SP0→SQ	1	Data valid
SP1_SQ_const_addr	SP1→SQ	36	Constant address load / predicate vector load (4 bits only)

			to the sequencer
SP1_SQ_valid	SP1→SQ	1	Data valid
SP2_SQ_const_addr	SP2→SQ	36	Constant address load / predicate vector load (4 bits only) to the sequencer
SP2_SQ_valid	SP2→SQ	1	Data valid
SP3_SQ_const_addr	SP3→SQ	36	Constant address load / predicate vector load (4 bits only) to the sequencer
SP3_SQ_valid	SP3→SQ	1	Data valid
SP0_SQ_data_type	SP→SQ	1	Data Type 0: Constant Load 1: Predicate Set

24.2.14 *SQ to SPx: constant broadcast*

Name	**Direction**	**Bits**	**Description**
SQ_SPx_const	SQ→SPx	128	Constant broadcast

24.2.15 *SP0 to SQ: Kill vector load*

Name	**Direction**	**Bits**	**Description**
SP0_SQ_kill_vect	SP0→SQ	4	Kill vector load
SP1_SQ_kill_vect	SP1→SQ	4	Kill vector load
SP2_SQ_kill_vect	SP2→SQ	4	Kill vector load
SP3_SQ_kill_vect	SP3→SQ	4	Kill vector load

24.2.16 *SQ to CP: RBBM bus*

Name	**Direction**	**Bits**	**Description**
SQ_RBB_rs	SQ→CP	1	Read Strobe
SQ_RBB_rd	SQ→CP	32	Read Data
SQ_RBBM_nrtrtr	SQ→CP	1	Optional
SQ_RBBM_rtr	SQ→CP	1	Real-Time (Optional)

24.2.17 *CP to SQ: RBBM bus*

Name	**Direction**	**Bits**	**Description**
rbbm_we	CP→SQ	1	Write Enable
rbbm_a	CP→SQ	15	Address -- Upper Extent is TBD (16:2)
rbbm_wd	CP→SQ	32	Data
rbbm_be	CP→SQ	4	Byte Enables
rbbm_re	CP→SQ	1	Read Enable
rbb_rs0	CP→SQ	1	Read Return Strobe 0
rbb_rs1	CP→SQ	1	Read Return Strobe 1
rbb_rd0	CP→SQ	32	Read Data 0
rbb_rd1	CP→SQ	32	Read Data 0
RBBM_SQ_soft_reset	CP→SQ	1	Soft Reset

24.2.18 *SQ to CP: State report*

Name	**Direction**	**Bits**	**Description**
SQ_CP_vs_event	SQ→CP	1	Vertex Shader Event
SQ_CP_vs_eventid	SQ→CP	2	Vertex Shader Event ID
SQ_CP_ps_event	SQ→CP	1	Pixel Shader Event
SQ_CP_ps_eventid	SQ→CP	2	Pixel Shader Event ID

eventid = 0 => *sEndOfState (i.e. VsEndOfState)
eventid = 1 => *sDone (i.e. VsDone)

So, the CP will assume the Vs is done with a state whenever it gets a pulse on the SQ_CP_vs_event and the SQ_CP_vs_eventid = 0.

24.3 Example of control flow program execution

We now provide some examples of execution to better illustrate the new design.

Given the program:

Alu 0
Alu 1
Tex 0
Tex 1
Alu 3 Serial
Alu 4
Tex 2
Alu 5
Alu 6 Serial
Tex 3
Alu 7
Alloc Position 1 buffer
Alu 8 Export
Tex 4
Alloc Parameter 3 buffers
Alu 9 Export 0
Tex 5
Alu 10 Serial Export 2
Alu 11 Export 1 End

Would be converted into the following CF instructions:

```
Execute Alu 0 Alu 0 Tex 0 Tex 0 Alu 1 Alu 0 Tex 0 Alu 0 Alu 1 Tex 0
Execute Alu 0
Alloc Position 1
Execute Alu 0 Tex 0
Alloc Param 3
Execute Alu 0 Tex 0 Alu 1 Alu 0 End
```

And the execution of this program would look like this:

Put thread in Vertex RS:

Control Flow Instruction Pointer (12 bits), (CFP)
Execution Count Marker (3 or 4 bits), (ECM)
Loop Iterators (4x9 bits), (LI)
Call return pointers (4x12 bits), (CRP)
Predicate Bits(4x64 bits), (PB)
Export ID (1 bit), (EXID)
GPR Base Ptr (8 bits), (GPR)
Export Base Ptr (7 bits), (EB)
Context Ptr (3 bits).(CPTR)
LOD correction bits (16x6 bits) (LOD)

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	0	0	0	0	0	0	0	0	0

Valid Thread (VALID)
Texture/ALU engine needed (TYPE)
Texture Reads are outstanding (PENDING)

Waiting on Texture Read to Complete (SERIAL)
Allocation Wait (2 bits) (ALLOC)
00 – No allocation needed
01 – Position export allocation needed (ordered export)
10 – Parameter or pixel export needed (ordered export)
11 – pass thru (out of order export)
Allocation Size (4 bits) (SIZE)
Position Allocated (POS_ALLOC)
First thread of a new context (FIRST)
Last (1 bit), (LAST)

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	0	0	0	0	0	1	0

Then the thread is picked up for the execution of the first control flow instruction:

```
Execute Alu 0 Alu 0 Tex 0 Tex 0 Alu 1 Alu 0 Tex 0 Alu 0 Alu 1 Tex 0
```

It executes the first two ALU instructions and goes back to the RS for a resource request change. Here is the state returned to the RS:

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	2	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	0	0	0	0	0	1	0

Then when the texture pipe frees up, the arbiter picks up the thread to issue the texture reads. The thread comes back in this state:

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	4	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	1	0	0	0	1	0

Because of the serial bit the arbiter must wait for the texture to return and clear the PENDING bit before it can pick the thread up. Lets say that the texture reads are complete, then the arbiter picks up the thread and returns it in this state:

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	6	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	0	0	0	0	0	1	0

Again the TP frees up, the arbiter picks up the thread and executes. It returns in this state:

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	7	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	0	0	0	1	0

Now, even if the texture has not returned we can still pick up the thread for ALU execution because the serial bit is not set. The thread will however come back to the RS for the second ALU instruction because it has the serial bit set.

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	8	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	1	0	0	0	1	0

As soon as the TP clears the pending bit the thread is picked up and returns:

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	9	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	0	0	0	0	0	1	0

Picked up by the TP and returns:

```
Execute Alu 0
```

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
1	0	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	0	0	0	1	0

Picked up by the ALU and returns (lets say the TP has not returned yet):

```
Alloc Position 1
```

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
2	0	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	01	1	0	1	0

If the SX has the place for the export, the SQ is going to allocate and pick up the thread for execution. It returns to the RS in this state:

```
Execute Alu 0 Tex 0
```

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
3	1	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	1	0	0	0	1	1	0

Now, since the TP has not returned yet, we must wait for it to return because we cannot issue multiple texture requests. The TP returns, clears the PENDING bit and we proceed:

```
    Alloc Param 3
```

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
4	0	0	0	0	1	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	10	3	1	1	0

Once again the SQ makes sure the SX has enough room in the Parameter cache before it can pick up this thread.

```
    Execute Alu 0 Tex 0 Alu 1 Alu 0 End
```

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
5	1	0	0	0	1	0	100	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	1	0	0	0	1	1	0

This executes on the TP and then returns:

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
5	2	0	0	0	1	0	100	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	1	0	0	1	1	1

Waits for the TP to return because of the textures reads are pending (and SERIAL in this case). Then executes and does not return to the RS because the LAST bit is set. This is the end of this thread and before dropping it on the floor, the SQ notifies the SX of export completion.

25. Open issues

Need to do some testing on the size of the register file as well as on the register file allocation method (dynamic VS static).

Saving power?

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 2015~~15~~ July, 200213 May, 2002	GEN-CXXXXX-REVA	1 of 54

Author: Laurent Lefebvre

Issue To:	**Copy No:**

R400 Sequencer Specification

SQ

Version 2.03~~2~~

Overview: This is an architectural specification for the R400 Sequencer block (SEQ). It provides an overview of the required capabilities and expected uses of the block. It also describes the block interfaces, internal sub-blocks, and provides internal state diagrams.

AUTOMATICALLY UPDATED FIELDS:
Document Location: C:\perforce\r400\doc_lib\design\blocks\sq\R400_Sequencer.doc
Current Intranet Search Title: R400 Sequencer Specification

APPROVALS	
Name/Dept	Signature/Date

Remarks:

THIS DOCUMENT CONTAINS CONFIDENTIAL INFORMATION THAT COULD BE SUBSTANTIALLY DETRIMENTAL TO THE INTEREST OF ATI TECHNOLOGIES INC. THROUGH UNAUTHORIZED USE OR DISCLOSURE.

ATI 2031
LG v. ATI
IPR2015-00326

ATI Ex. 2119
IPR2023-00922

Table Of Contents

1. OVERVIEW .... **9~~7~~**
1.1 Top Level Block Diagram .... 11~~9~~
1.2 Data Flow graph (SP) .... 12~~10~~
1.3 Control Graph .... 13~~11~~
2. INTERPOLATED DATA BUS .... **13~~11~~**
3. INSTRUCTION STORE .... **16~~14~~**
4. SEQUENCER INSTRUCTIONS .... **16~~14~~**
5. CONSTANT STORES .... **16~~14~~**
5.1 Memory organizations .... 16~~14~~
5.2 Management of the Control Flow Constants .... 17~~15~~
5.3 Management of the re-mapping tables .... 17~~15~~
5.3.1 R400 Constant management .... 17~~15~~
5.3.2 Proposal for R400LE constant management .... 17~~15~~
5.3.3 Dirty bits .... 19~~17~~
5.3.4 Free List Block .... 19~~17~~
5.3.5 De-allocate Block .... 20~~18~~
5.3.6 Operation of Incremental model .... 20~~18~~
5.4 Constant Store Indexing .... 20~~18~~
5.5 Real Time Commands .... 21~~19~~
5.6 Constant Waterfalling .... 21~~19~~
6. LOOPING AND BRANCHES .... **22~~20~~**
6.1 The controlling state. .... 22~~20~~
6.2 The Control Flow Program .... 22~~20~~
6.2.1 Control flow instructions table .... 23~~21~~
6.3 Implementation .... 25~~23~~
6.4 Data dependant predicate instructions .... 26~~24~~
6.5 HW Detection of PV,PS .... 27~~25~~
6.6 Register file indexing .... 27~~25~~
6.7 Debugging the Shaders .... 28~~25~~
6.7.1 Method 1: Debugging registers .... 28~~26~~
6.7.2 Method 2: Exporting the values in the GPRs .... 28~~26~~
7. PIXEL KILL MASK .... **28~~26~~**
8. MULTIPASS VERTEX SHADERS (HOS) .... **28~~26~~**
9. REGISTER FILE ALLOCATION .... **29~~26~~**
10. FETCH ARBITRATION .... **30~~28~~**
11. ALU ARBITRATION .... **30~~28~~**
12. HANDLING STALLS .... **31~~29~~**
13. CONTENT OF THE RESERVATION STATION FIFOS .... **31~~29~~**
14. THE OUTPUT FILE .... **31~~29~~**
15. IJ FORMAT .... **31~~29~~**
15.1 Interpolation of constant attributes .... 32~~29~~
16. STAGING REGISTERS .... **32~~30~~**

17. THE PARAMETER CACHE .... **34~~31~~**
17.1 Export restrictions .... 34~~32~~
17.1.1 Pixel exports: .... 34~~32~~
17.1.2 Vertex exports: .... 34~~32~~
17.1.3 Pass thru exports: .... 34~~32~~
17.2 Arbitration restrictions .... 34~~32~~
18. EXPORT TYPES .... **35~~32~~**
18.1 Vertex Shading .... 35~~32~~
18.2 Pixel Shading .... 35~~33~~
19. SPECIAL INTERPOLATION MODES .... **35~~33~~**
19.1 Real time commands .... 35~~33~~
19.2 Sprites/ XY screen coordinates/ FB information .... 36~~33~~
19.3 Auto generated counters .... 36~~34~~
19.3.1 Vertex shaders .... 36~~34~~
19.3.2 Pixel shaders .... 36~~34~~
20. STATE MANAGEMENT .... **37~~34~~**
20.1 Parameter cache synchronization .... 37~~34~~
21. XY ADDRESS IMPORTS .... **37~~35~~**
21.1 Vertex indexes imports .... 37~~35~~
22. REGISTERS .... **37~~35~~**
22.1 Control .... 37~~35~~
22.2 Context .... 38~~35~~
23. DEBUG REGISTERS .... **39~~35~~**
23.1 Context .... 39~~35~~
23.2 Control .... 39~~35~~
24. INTERFACES .... **39~~35~~**
24.1 External Interfaces .... 39~~35~~
24.2 SC to SP Interfaces .... 39~~35~~
24.2.1 SC_SP# .... 39~~35~~
24.2.2 SC_SQ .... 40~~36~~
24.2.3 SQ to SX: Interpolator bus .... 42~~38~~
24.2.4 SQ to SP: Staging Register Data .... 42~~38~~
24.2.5 VGT to SQ : Vertex interface .... 42~~38~~
24.2.6 SQ to SX: Control bus .... 45~~41~~
24.2.7 SX to SQ : Output file control .... 45~~41~~
24.2.8 SQ to TP: Control bus .... 46~~42~~
24.2.9 TP to SQ: Texture stall .... 46~~42~~
24.2.10 SQ to SP: Texture stall .... 47~~43~~
24.2.11 SQ to SP: GPR and auto counter .... 47~~43~~
24.2.12 SQ to SPx: Instructions .... 48~~44~~
24.2.13 SP to SQ: Constant address load/ Predicate Set .... 49~~44~~
24.2.14 SQ to SPx: constant broadcast .... 49~~45~~

24.2.15 SP0 to SQ: Kill vector load .... 49~~45~~
24.2.16 SQ to CP: RBBM bus .... 49~~45~~
24.2.17 CP to SQ: RBBM bus .... 49~~45~~
24.2.18 SQ to CP: State report .... 50~~45~~
24.3 Example of control flow program execution .... 50~~46~~
25. OPEN ISSUES .... 54~~50~~
~~**1. OVERVIEW .... 7**~~
~~1.1 Top Level Block Diagram .... 9~~
~~1.2 Data Flow graph (SP) .... 10~~
~~1.3 Control Graph .... 11~~
~~**2. INTERPOLATED DATA BUS .... 11**~~
~~**3. INSTRUCTION STORE .... 14**~~
~~**4. SEQUENCER INSTRUCTIONS .... 14**~~
~~**5. CONSTANT STORES .... 14**~~
~~5.1 Memory organizations .... 14~~
~~5.2 Management of the Control Flow Constants .... 15~~
~~5.3 Management of the re-mapping tables .... 15~~
~~5.3.1 R400 Constant management .... 15~~
~~5.3.2 Proposal for R400LE constant management .... 15~~
~~5.3.3 Dirty bits .... 17~~
~~5.3.4 Free List Block .... 17~~
~~5.3.5 De-allocate Block .... 18~~
~~5.3.6 Operation of Incremental model .... 18~~
~~5.4 Constant Store Indexing .... 18~~
~~5.5 Real Time Commands .... 19~~
~~5.6 Constant Waterfalling .... 19~~
~~**6. LOOPING AND BRANCHES .... 20**~~
~~6.1 The controlling state. .... 20~~
~~6.2 The Control Flow Program .... 20~~
~~6.2.1 Control flow instructions table .... 21~~
~~6.3 Implementation .... 22~~
~~6.4 Data dependant predicate instructions .... 24~~
~~6.5 HW Detection of PV,PS .... 24~~
~~6.6 Register file indexing .... 25~~
~~6.7 Debugging the Shaders .... 25~~
~~6.7.1 Method 1: Debugging registers .... 25~~
~~6.7.2 Method 2: Exporting the values in the GPRs .... 26~~
~~**7. PIXEL KILL MASK .... 26**~~
~~**8. MULTIPASS VERTEX SHADERS (HOS) .... 26**~~
~~**9. REGISTER FILE ALLOCATION .... 26**~~
~~**10. FETCH ARBITRATION .... 27**~~
~~**11. ALU ARBITRATION .... 27**~~
~~**12. HANDLING STALLS .... 28**~~
~~**13. CONTENT OF THE RESERVATION STATION FIFOS .... 28**~~
~~**14. THE OUTPUT FILE .... 28**~~

~~15. IJ FORMAT .... 28~~
~~15.1 Interpolation of constant attributes .... 29~~
~~16. STAGING REGISTERS .... 29~~
~~17. THE PARAMETER CACHE .... 31~~
~~17.1 Export restrictions .... 31~~
~~17.1.1 Pixel exports: .... 31~~
~~17.1.2 Vertex exports: .... 31~~
~~17.1.3 Pass thru exports: .... 31~~
~~17.2 Arbitration restrictions .... 31~~
~~18. EXPORT TYPES .... 32~~
~~18.1 Vertex Shading .... 32~~
~~18.2 Pixel Shading .... 32~~
~~19. SPECIAL INTERPOLATION MODES .... 32~~
~~19.1 Real time commands .... 32~~
~~19.2 Sprites/ XY screen coordinates/ FB information .... 33~~
~~19.3 Auto generated counters .... 33~~
~~19.3.1 Vertex shaders .... 33~~
~~19.3.2 Pixel shaders .... 33~~
~~20. STATE MANAGEMENT .... 34~~
~~20.1 Parameter cache synchronization .... 34~~
~~21. XY ADDRESS IMPORTS .... 34~~
~~21.1 Vertex indexes imports .... 34~~
~~22. REGISTERS .... 35~~
~~22.1 Control .... 35~~
~~22.2 Context .... 35~~
~~23. DEBUG REGISTERS .... 36~~
~~23.1 Context .... 36~~
~~23.2 Control .... 36~~
~~24. INTERFACES .... 36~~
~~24.1 External Interfaces .... 36~~
~~24.2 SC to SP Interfaces .... 36~~
~~24.2.1 SC_SP# .... 36~~
~~24.2.2 SC_SQ .... 37~~
~~24.2.3 SQ to SX: Interpolator bus .... 39~~
~~24.2.4 SQ to SP: Staging Register Data .... 39~~
~~24.2.5 VGT to SQ : Vertex interface .... 39~~
~~24.2.6 SQ to SX: Control bus .... 42~~
~~24.2.7 SX to SQ : Output file control .... 42~~
~~24.2.8 SQ to TP: Control bus .... 43~~
~~24.2.9 TP to SQ: Texture stall .... 43~~
~~24.2.10 SQ to SP: Texture stall .... 44~~
~~24.2.11 SQ to SP: GPR and auto counter .... 44~~
~~24.2.12 SQ to SPx: Instructions .... 45~~

~~24.2.13 SP to SQ: Constant address load/ Predicate Set.. 45~~
~~24.2.14 SQ to SPx: constant broadcast.. 46~~
~~24.2.15 SP0 to SQ: Kill vector load... 46~~
~~24.2.16 SQ to CP: RBBM bus... 46~~
~~24.2.17 CP to SQ: RBBM bus... 46~~
~~24.2.18 SQ to CP: State report.. 46~~
~~24.3 Example of control flow program execution.. 46~~
~~**25. OPEN ISSUES.. 51**~~

Revision Changes:

Rev 0.1 (Laurent Lefebvre) Date: May 7, 2001	First draft.
Rev 0.2 (Laurent Lefebvre) Date : July 9, 2001	Changed the interfaces to reflect the changes in the SP. Added some details in the arbitration section.
Rev 0.3 (Laurent Lefebvre) Date : August 6, 2001	Reviewed the Sequencer spec after the meeting on August 3, 2001.
Rev 0.4 (Laurent Lefebvre) Date : August 24, 2001	Added the dynamic allocation method for register file and an example (written in part by Vic) of the flow of pixels/vertices in the sequencer.
Rev 0.5 (Laurent Lefebvre) Date : September 7, 2001	Added timing diagrams (Vic)
Rev 0.6 (Laurent Lefebvre) Date : September 24, 2001	Changed the spec to reflect the new R400 architecture. Added interfaces.
Rev 0.7 (Laurent Lefebvre) Date : October 5, 2001	Added constant store management, instruction store management, control flow management and data dependant predication.
Rev 0.8 (Laurent Lefebvre) Date : October 8, 2001	Changed the control flow method to be more flexible. Also updated the external interfaces.
Rev 0.9 (Laurent Lefebvre) Date : October 17, 2001	Incorporated changes made in the 10/18/01 control flow meeting. Added a NOP instruction, removed the conditional_execute_or_jump. Added debug registers.
Rev 1.0 (Laurent Lefebvre) Date : October 19, 2001	Refined interfaces to RB. Added state registers.
Rev 1.1 (Laurent Lefebvre) Date : October 26, 2001	Added SEQ→SP0 interfaces. Changed delta precision. Changed VGT→SP0 interface. Debug Methods added.
Rev 1.2 (Laurent Lefebvre) Date : November 16, 2001	Interfaces greatly refined. Cleaned up the spec.
Rev 1.3 (Laurent Lefebvre) Date : November 26, 2001	Added the different interpolation modes.
Rev 1.4 (Laurent Lefebvre) Date : December 6, 2001	Added the auto incrementing counters. Changed the VGT→SQ interface. Added content on constant management. Updated GPRs.
Rev 1.5 (Laurent Lefebvre) Date : December 11, 2001	Removed from the spec all interfaces that weren't directly tied to the SQ. Added explanations on constant management. Added PA→SQ synchronization fields and explanation.
Rev 1.6 (Laurent Lefebvre) Date : January 7, 2002	Added more details on the staging register. Added detail about the parameter caches. Changed the call instruction to a Conditionnal_call instruction. Added details on constant management and updated the diagram.
Rev 1.7 (Laurent Lefebvre) Date : February 4, 2002	Added Real Time parameter control in the SX interface. Updated the control flow section.
Rev 1.8 (Laurent Lefebvre) Date : March 4, 2002	New interfaces to the SX block. Added the end of clause modifier, removed the end of clause instructions.
Rev 1.9 (Laurent Lefebvre) Date : March 18, 2002	Rearangement of the CF instruction bits in order to ensure byte alignement.
Rev 1.10 (Laurent Lefebvre) Date : March 25, 2002	Updated the interfaces and added a section on exporting rules.
Rev 1.11 (Laurent Lefebvre) Date : April 19, 2002	Added CP state report interface. Last version of the spec with the old control flow scheme
Rev 2.0 (Laurent Lefebvre) Date : April 19, 2002	New control flow scheme

Rev 2.01 (Laurent Lefebvre) Date : May 2, 2002	Changed slightly the control flow instructions to allow force jumps and calls.
Rev 2.02 (Laurent Lefebvre) Date : May 13, 2002	Updated the Opcodes. Added type field to the constant/pred interface. Added Last field to the SQ→SP instruction load interface.
Rev 2.03 (Laurent Lefebvre) Date : July 15, 2002	SP interface updated to include predication optimizations. Added the predicate no stall instructions,

1. Overview

The sequencer chooses two ALU threads and a fetch hread to execute, and executes all of the instructions in a block before looking for a new clause of the same type. Two ALU threads are executed interleaved to hide the ALU latency. The arbitrator will give priority to older threads. There are two separate reservation stations, one for pixel vectors and one for vertices vectors. This way a pixel can pass a vertex and a vertex can pass a pixel.

To support the shader pipe the sequencer also contains the shader instruction cache, constant store, control flow constants and texture state. The four shader pipes also execute the same instruction thus there is only one sequencer for the whole chip.

The sequencer first arbitrates between vectors of 64 vertices that arrive directly from primitive assembly and vectors of 16 quads (64 pixels) that are generated in the scan converter.

The vertex or pixel program specifies how many GPRs it needs to execute. The sequencer will not start the next vector until the needed space is available in the GPRs.

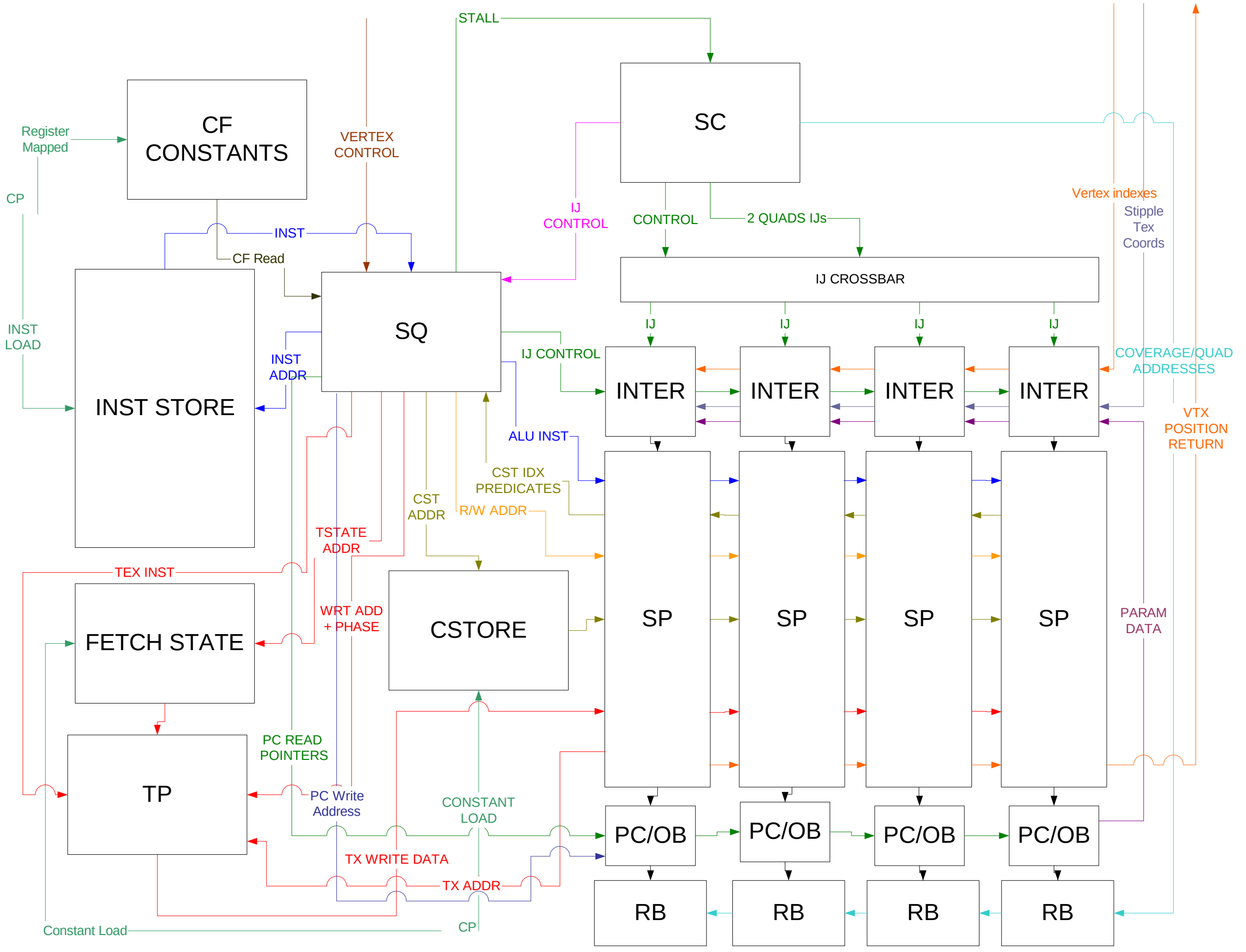


Figure 1: General Sequencer overview

1.1 Top Level Block Diagram

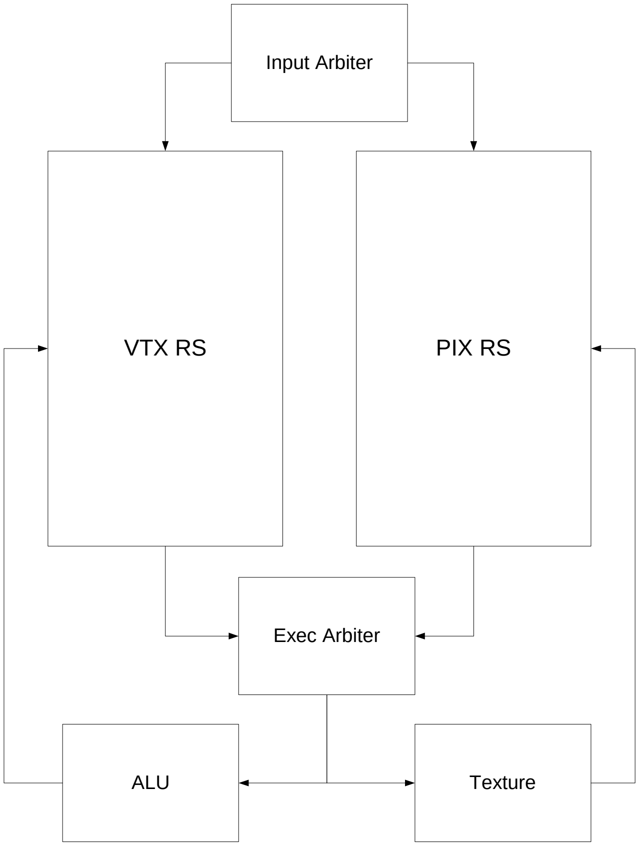


Figure 2: Reservation stations and arbiters

Under this new scheme, the sequencer (SQ) will only use one global state management machine per vector type (pixel, vertex) that we call the reservation station (RS).

1.2 Data Flow graph (SP)

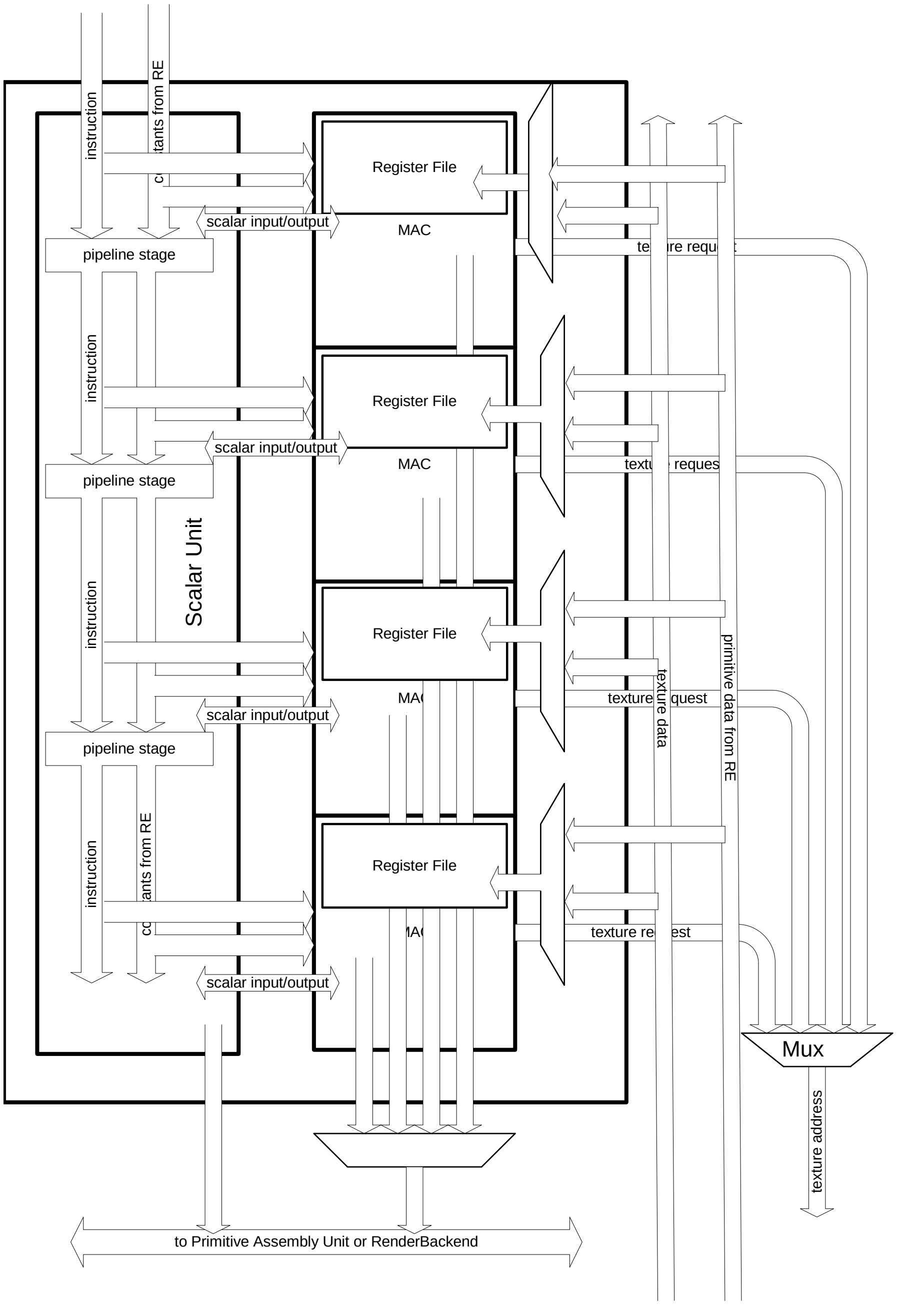


Figure 3: The shader Pipe

The gray area represents blocks that are replicated 4 times per shader pipe (16 times on the overall chip).

1.3 Control Graph

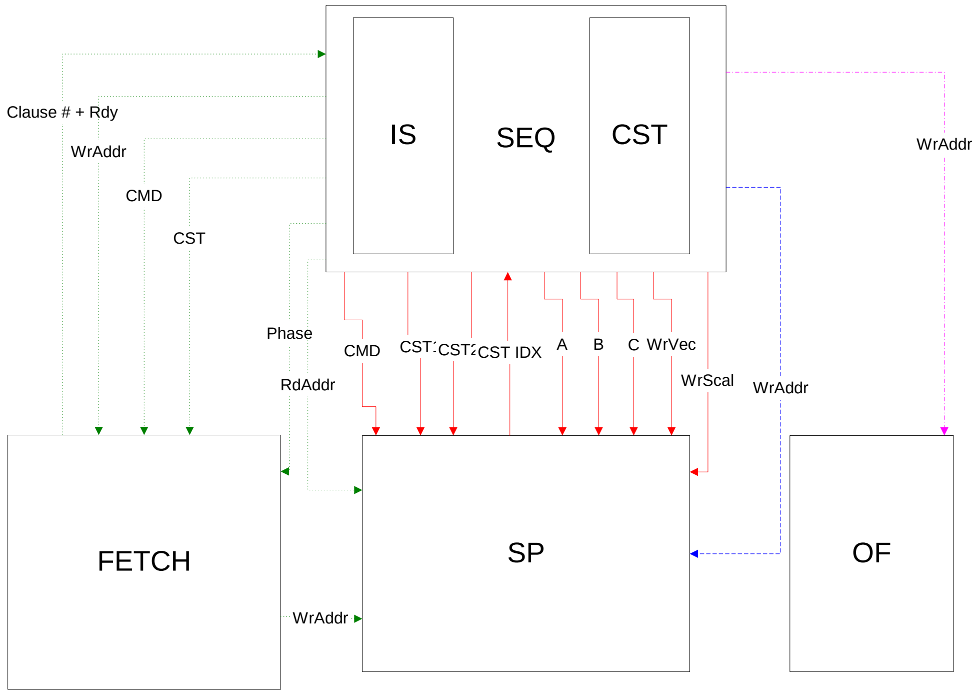


Figure 4: Sequencer Control interfaces

In green is represented the Fetch control interface, in red the ALU control interface, in blue the Interpolated/Vector control interface and in purple is the output file control interface.

2. Interpolated data bus

The interpolators contain an IJ buffer to pack the information as much as possible before writing it to the register file.

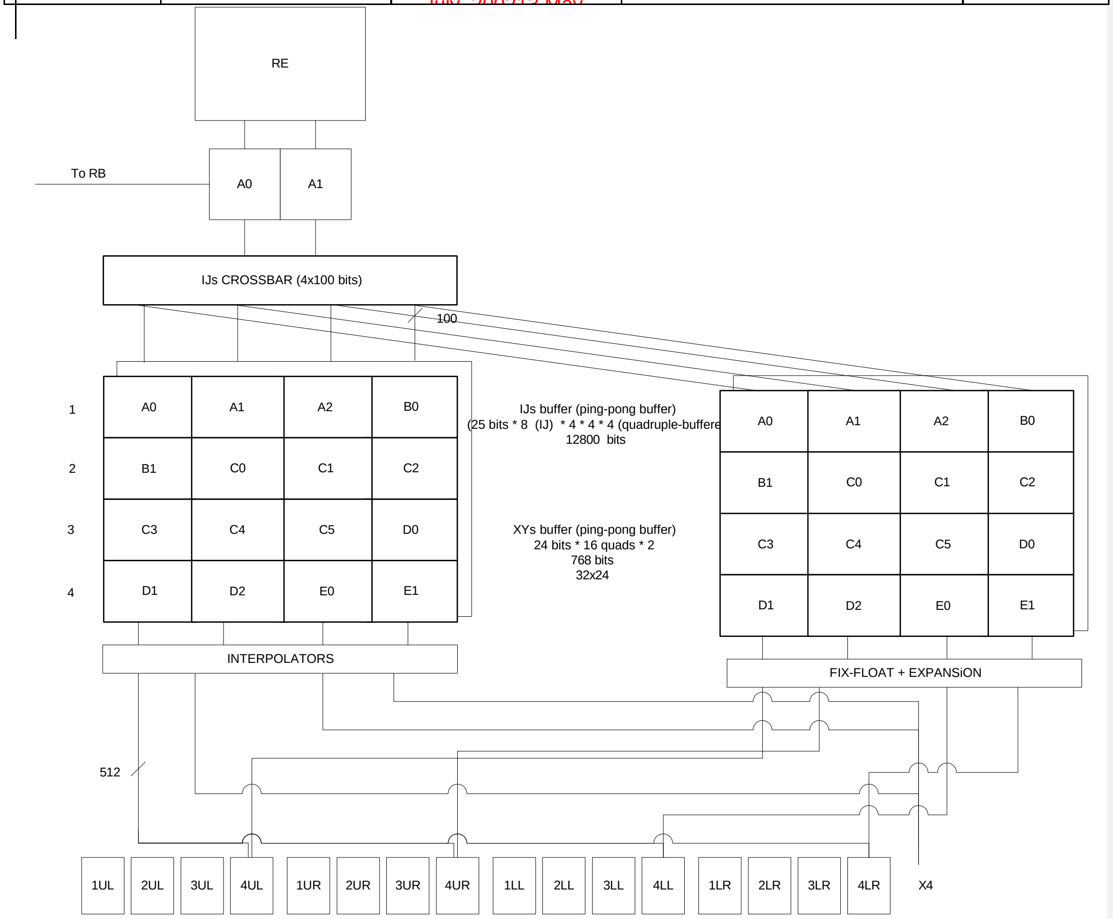


Figure 5: Interpolation buffers

WRITES

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21	T22	T23
SP 0	A0	A0	XY A0	B1	B1	XY B1	C3	C3	XY C3				D1	D1	XY D1									
SP 1	A1	A1	XY A1				C0	C0	XY C0	C4	C4	XY C4	D2	D2	XY D2									
SP 2	A2	A2	XY A2				C1	C1	XY C1	C5	C5	XY C5				E0	E0	XY E0						
SP 3				B0	B0	XY B0	C2	C2	XY C2				D0	D0	XY D0	E1	E1	XY E1						

READS

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21	T22	T23
SP 0	XY 0-3	XY 16-19	XY 32-35	XY 48-51	A0	B1	C3	D1					A0	B1	C3	D1					V 0-3	V 16-19	V 32-35	V 48-51
SP 1	XY 4-7	XY 20-23	XY 36-39	XY 52-55	A1		C4	D2		C0			A1		C4	D2		C0			V 4-7	V 20-23	V 36-39	V 52-55
SP 2	XY 8-11	XY 24-27	XY 40-43	XY 56-59	A2		C5			C1		E0	A2		C5			C1		E0	V 8-11	V 24-27	V 40-43	V 56-59
SP 3	XY 12-15	XY 28-31	XY 44-47	XY 60-63					B0	C2	D0	E1					B0	C2	D0	E1	V 12-15	V 28-31	V 44-47	V 60-63
		XY						P1									P2						VTX	

Figure 6: Interpolation timing diagram

Above is an example of a tile the sequencer might receive from the SC. The write side is how the data get stacked into the XY and IJ buffers, the read side is how the data is passed to the GPRs. The IJ information is packed in the IJ buffer 4 quads at a time or two clocks. The sequencer allows at any given time as many as four quads to interpolate a parameter. They all have to come from the same primitive. Then the sequencer controls the write mask to the GPRs to write the valid data in.

3. Instruction Store

There is going to be only one instruction store for the whole chip. It will contain 4096 instructions of 96 bits each.

It is likely to be a 1 port memory; we use 1 clock to load the ALU instruction, 1 clocks to load the Fetch instruction, 1 clock to load 2 control flow instructions and 1 clock to write instructions.

The instruction store is loaded by the CP thru the register mapped registers.

The VS_BASE and PS_BASE context registers are used to specify for each context where its shader is in the instruction memory.

For the Real time commands the story is quite the same but for some small differences. There are no wrap-around points for real time so the driver must be careful not to overwrite regular shader data. The shared code (shared subroutines) uses the same path as real time.

4. Sequencer Instructions

All control flow instructions and move instructions are handled by the sequencer only. The ALUs will perform NOPs during this time (MOV PV,PV, PS,PS) if they have nothing else to do.

5. Constant Stores

5.1 Memory organizations

A likely size for the ALU constant store is 1024x128 bits. The read BW from the ALU constant store is 128 bits/clock and the write bandwidth is 32 bits/clock (directed by the CP bus size not by memory ports).

The maximum logical size of the constant store for a given shader is 256 constants. Or 512 for the pixel/vertex shader pair. The size of the re-mapping table is 128 lines (each line addresses 4 constants). The write granularity is 4 constants or 512 bits. It takes 16 clocks to write the four constants. Real time requires 256 lines in the physical memory (this is physically register mapped).

The texture state is also kept in a similar memory. The size of this memory is 320x96 bits (128 texture states for regular mode, 32 states for RT). The memory thus holds 128 texture states (192 bits per state). The logical size exposes 32 different states total, which are going to be shared between the pixel and the vertex shader. The size of the re-mapping table to for the texture state memory is 32 lines (each line addresses 1 texture state lines in the real memory). The CP write granularity is 1 texture state lines (or 192 bits). The driver sends 512 bits but the CP ignores the top 320 bits. It thus takes 6 clocks to write the texture state. Real time requires 32 lines in the physical memory (this is physically register mapped).

The control flow constant memory doesn't sit behind a renaming table. It is register mapped and thus the driver must reload its content each time there is a change in the control flow constants. Its size is 320*32 because it must hold 8 copies of the 32 dwords of control flow constants and the loop construct constants must be aligned.

The constant re-mapping tables for texture state and ALU constants are logically register mapped for regular mode and physically register mapped for RT operation.

5.2 Management of the Control Flow Constants

The control flow constants are register mapped, thus the CP writes to the according register to set the constant, the SQ decodes the address and writes to the block pointed by its current base pointer (CF_WR_BASE). On the read side, one level of indirection is used. A register (SQ_CONTEXT_MISC.CF_RD_BASE) keeps the current base pointer to the control flow block. This register is copied whenever there is a state change. Should the CP write to CF after the state change, the base register is updated with the (current pointer number +1)% number of states. This way, if the CP doesn't write to CF the state is going to use the previous CF constants.

5.3 Management of the re-mapping tables

5.3.1 *R400 Constant management*

The sequencer is responsible to manage two re-mapping tables (one for the constant store and one for the texture state). On a state change (by the driver), the sequencer will broadside copy the contents of its re-mapping tables to a new one. We have 8 different re-mapping tables we can use concurrently.

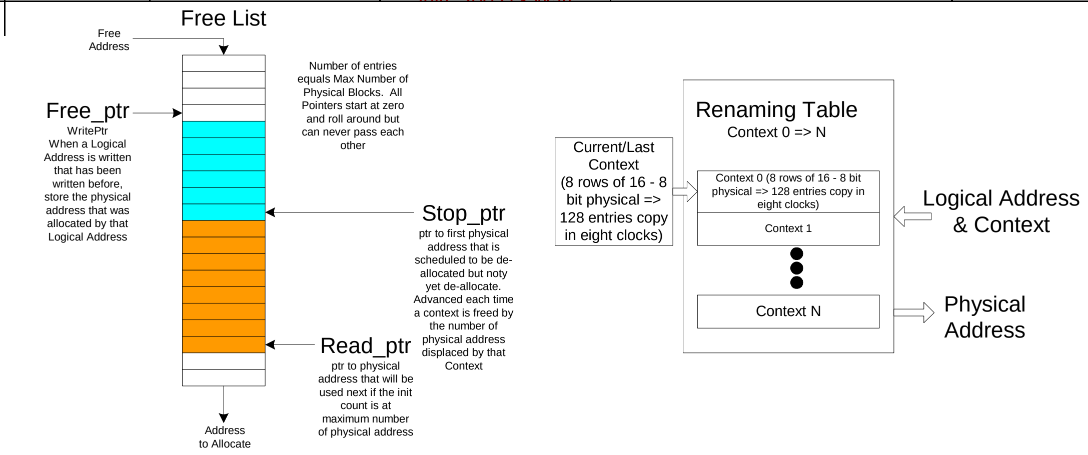
The constant memory update will be incremental, the driver only need to update the constants that actually changed between the two state changes.

For this model to work in its simplest form, the requirement is that the physical memory MUST be at least twice as large as the logical address space + the space allocated for Real Time. In our case, since the logical address space is 512 and the reserved RT space can be up to 256 entries, the memory must be of sizes 1280 and above. Similarly the size of the texture store must be of 32*2+32 = 96 entries and above.

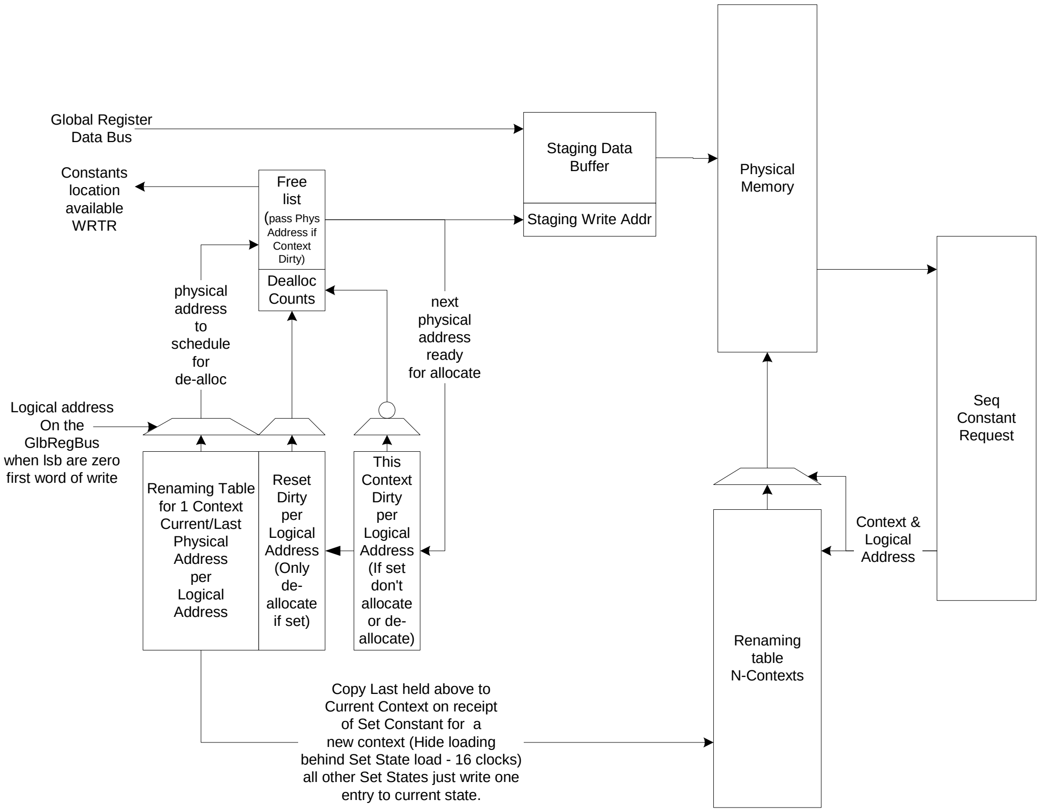
5.3.2 *Proposal for R400LE constant management*

To make this scheme work with only 512+256 = 768 entries, upon reception of a CONTROL packet of state + 1, the sequencer would check for SQ_IDLE and PA_IDLE and if both are idle will erase the content of state to replace it with the new state (this is depicted in Figure 8: De-allocation mechanism~~Figure 8: De-allocation mechanismFigure 8: De-allocation mechanism~~). Note that in the case a state is cleared a value of 0 is written to the corresponding de-allocation counter location so that when the SQ is going to report a state change, nothing will be de-allocated upon the first report.

The second path sets all context dirty bits that were used in the current state to 1 (thus allowing the new state to reuse these physical addresses if needed).



Free List
Free Address
Free_ptr
WritePtr
When a Logical Address is written that has been written before, store the physical address that was allocated by that Logical Address
Number of entries equals Max Number of Physical Blocks. All Pointers start at zero and roll around but can never pass each other
Stop_ptr
ptr to first physical address that is scheduled to be de-allocated but noty yet de-allocate. Advanced each time a context is freed by the number of physical address displaced by that Context
Read_ptr
ptr to physical address that will be used next if the init count is at maximum number of physical address
Address to Allocate
Current/Last Context (8 rows of 16 - 8 bit physical => 128 entries copy in eight clocks)
Renaming Table
Context 0 => N
Context 0 (8 rows of 16 - 8 bit physical => 128 entries copy in eight clocks)
Context 1
Context N
Logical Address & Context
Physical Address



Global Register Data Bus
Staging Data Buffer
Staging Write Addr
Physical Memory
Constants location available WRTR
Free list (pass Phys Address if Context Dirty)
Dealloc Counts
physical address to schedule for de-alloc
next physical address ready for allocate
Seq Constant Request
Logical address On the GlbRegBus when lsb are zero first word of write
Renaming Table for 1 Context Current/Last Physical Address per Logical Address
Reset Dirty per Logical Address (Only de-allocate if set)
This Context Dirty per Logical Address (If set don't allocate or de-allocate)
Context & Logical Address
Renaming table N-Contexts
Copy Last held above to Current Context on receipt of Set Constant for a new context (Hide loading behind Set State load - 16 clocks) all other Set States just write one entry to current state.

Figure 7: Constant management

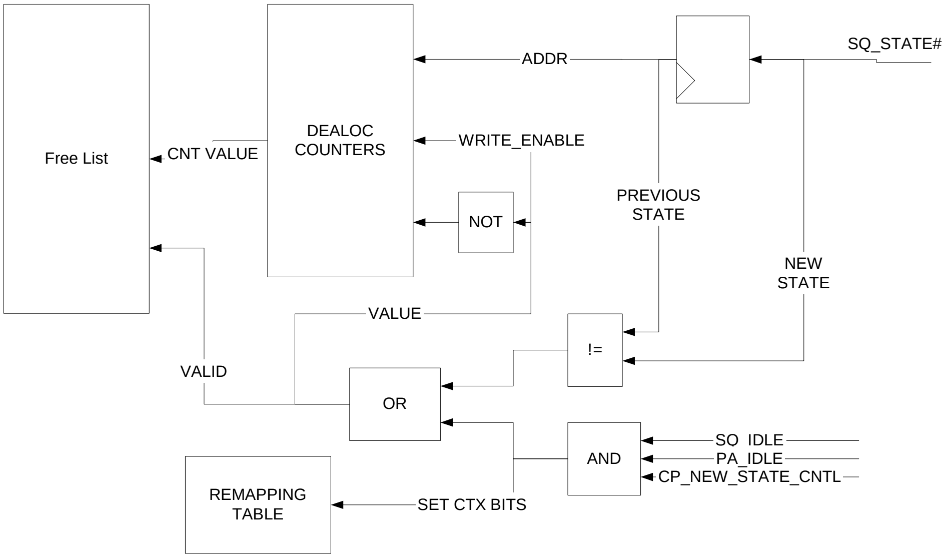


Figure 8: De-allocation mechanism for R400LE

5.3.3 *Dirty bits*

Two sets of dirty bits will be maintained per logical address. The first one will be set to zero on reset and set when the logical address is addressed. The second one will be set to zero whenever a new context is written and set for each address written while in this context. The reset dirty is not set, then writing to that logical address will not require de-allocation of whatever address stored in the renaming table. If it is set and the context dirty is not set, then the physical address store needs to be de-allocated and a new physical address is necessary to store the incoming data. If they are both set, then the data will be written into the physical address held in the renaming for the current logical address. No de-allocation or allocation takes place. This will happen when the driver does a set constant twice to the same logical address between context changes. NOTE: It is important to detect and prevent this, failure to do it will allow multiple writes to allocate all physical memory and thus hang because a context will not fit for rendering to start and thus free up space.

5.3.4 *Free List Block*

A free list block that would consist of a counter (called the IFC or Initial Free Counter) that would reset to zero and incremented every time a chunk of physical memory is used until they have all been used once. This counter would be checked each time a physical block is needed, and if the original ones have not been used up, us a new one, else check the free list for an available physical block address. The count is the physical address for when getting a chunk from the counter.

Storage of a free list big enough to store all physical block addresses.

Maintain three pointers for the free list that are reset to zero. The first one we will call write_ptr. This pointer will identify the next location to write the physical address of a block to be de-allocated. Note: we can never free more physical memory locations than we have. Once recording address the pointer will be incremented to walk the free list like a ring.

The second pointer will be called stop_ptr. The stop_ptr pointer will be advanced by the number of address chunks de-allocates when a context finishes. The address between the stop_ptr and write_ptr cannot be reused because they are still in use. But as soon as the context using then is dismissed the stop_ptr will be advanced.

The third pointer will be called read_ptr. This pointer will point will point to the next address that can be used for allocation as long as the read_ptr does not equal the stop_ptr and the IFC is at its maximum count.

5.3.5 *De-allocate Block*

This block will maintain a free physical address block count for each context. While in current context, a count shall be maintained specifying how many blocks were written into the free list at the write_ptr pointer. This count will be reset upon reset or when this context is active on the back and different than the previous context. It is actually a count of blocks in the previous context that will no longer be used. This count will be used to advance the write_ptr pointer to make available the set of physical blocks freed when the previous context was done. This allows the discard or de-allocation of any number of blocks in one clock.

5.3.6 *Operation of Incremental model*

The basic operation of the model would start with the write_ptr, stop_ptr, read_ptr pointers in the free list set to zero and the free list counter is set to zero. Also all the dirty bits and the previous context will be initialized to zero. When the first set constants happen, the reset dirty bit will not be set, so we will allocate a physical location from the free list counter because its not at the max value. The data will be written into physical address zero. Both the additional copy of the renaming table and the context zeros of the big renaming table will be updated for the logical address that was written by set start with physical address of 0. This process will be repeated for any logical address that are not dirty until the context changes. If a logical address is hit that has its dirty bits set while in the same context, both dirty bits would be set, so the new data will be over-written to the last physical address assigned for this logical address. When the first draw command of the context is detected, the previous context stored in the additional renaming table will be copied to the larger renaming table in the current (new) context location. Then the set constant logical address with be loaded with a new physical address during the copy and if the reset dirty was set, the physical address it replaced in the renaming table would be entered at the write_ptr pointer location on the free list and the write_ptr will be incremented. The de-allocation counter for the previous context (eight) will be incremented. This as set states come in for this context one of the following will happen:

1.) No dirty bits are set for the logical address being updated. A line will be allocated of the free-list counter or the free list at read_ptr pointer if read_ptr != to stop_ptr .
2.) Reset dirty set and Context dirty not set. A new physical address is allocated, the physical address in the renaming table is put on the free list at write_ptr and it is incremented along with the de-allocate counter for the last context.
3.) Context dirty is set then the data will be written into the physical address specified by the logical address.

This process will continue as long as set states arrive. This block will provide backpressure to the CP whenever he has not free list entries available (counter at max and stop_ptr == read_ptr). The command stream will keep a count of contexts of constants in use and prevent more than max constants contexts from being sent.

Whenever a draw packet arrives, the content of the re-mapping table is written to the correct re-mapping table for the context number. Also if the next context uses less constants than the current one all exceeding lines are moved to the free list to be de-allocated later. This happens in parallel with the writing of the re-mapping table to the correct memory.

Now preferable when the constant context leaves the last ALU clause it will be sent to this block and compared with the previous context that left. (Init to zero) If they differ than the older context will no longer be referenced and thus can be de-allocated in the physical memory. This is accomplished by adding the number of blocks freed this context to the stop_ptr pointer. This will make all the physical addresses used by this context available to the read_ptr allocate pointer for future allocation.

This device allows representation of multiple contexts of constants data with N copies of the logical address space. It also allows the second context to be represented as the first set plus some new additional data by just storing the delta's. It allows memory to be efficiently used and when the constants updates are small it can store multiple context. However, if the updates are large, less contexts will be stored and potentially performance will be degraded. Although it will still perform as well as a ring could in this case.

5.4 Constant Store Indexing

In order to do constant store indexing, the sequencer must be loaded first with the indexes (that come from the GPRs). There are 144 wires from the exit of the SP to the sequencer (9 bits pointers x 16 vertexes/clock). Since the data must pass thru the Shader pipe for the float to fixed conversion, there is a latency of 4 clocks (1 instruction)

between the time the sequencer is loaded and the time one can index into the constant store. The assembly will look like this

```
MOVA  R1.X,R2.X       // Loads the sequencer with the content of R2.X, also copies the content of R2.X into R1.X
NOP                   // latency of the float to fixed conversion
ADD   R3,R4,C0[R2.X]// Uses the state from the sequencer to add R4 to C0[R2.X] into R3
```

Note that we don't really care about what is in the brackets because we use the state from the MOVA instruction. R2.X is just written again for the sake of simplicity and coherency.

The storage needed in the sequencer in order to support this feature is 2*64*9 bits = 1152 bits.

5.5 Real Time Commands

The real time commands constants are written by the CP using the register mapped registers allocated for RT. It works is the same way than when dealing with regular constant loads BUT in this case the CP is not sending a logical address but rather a physical address and the reads are not passing thru the re-mapping table but are directly read from the memory. The boundary between the two zones is defined by the CONST_EO_RT control register. Similarly, for the fetch state, the boundary between the two zones is defined by the TSTATE_EO_RT control register.

5.6 Constant Waterfalling

In order to have a reasonable performance in the case of constant store indexing using the address register, we are going to have the possibility of using the physical memory port for read only. This way we can read 1 constant per clock and thus have a worst-case waterfall mode of 1 vertex per clock. There is a small synchronization issue related with this as we need for the SQ to make sure that the constants where actually written to memory (not only sent to the sequencer) before it can allow the first vector of pixels or vertices of the state to go thru the ALUs. To do so, the sequencer keeps 8 bits (one per render state) and sets the bits whenever the last render state is written to memory and clears the bit whenever a state is freed.

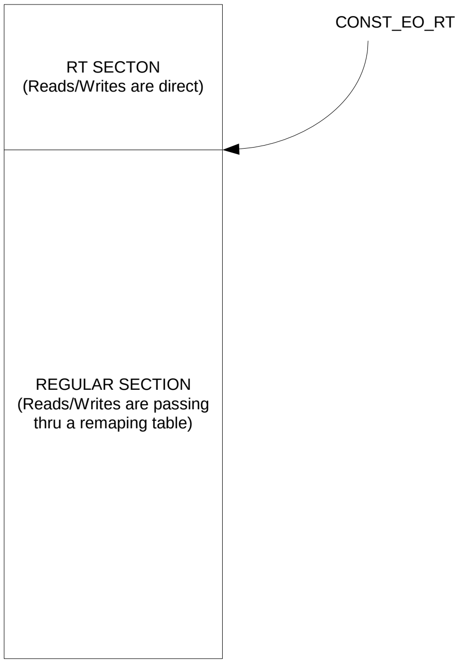

CONST_EO_RT
RT SECTON
(Reads/Writes are direct)
REGULAR SECTION
(Reads/Writes are passing thru a remaping table)

Figure 9: The ~~instruction~~ Constant store

6. Looping and Branches

Loops and branches are planned to be supported and will have to be dealt with at the sequencer level. We plan on supporting constant loops and branches using a control program.

6.1 The controlling state.

The R400 controling state consists of:

Boolean[256:0]
Loop_count[7:0][31:0]
Loop_Start[7:0][31:0]
Loop_Step[7:0][31:0]

That is 256 Booleans and 32 loops.

We have a stack of 4 elements for nested calls of subroutines and 4 loop counters to allow for nested loops.

This state is available on a per shader program basis.

6.2 The Control Flow Program

We'd like to be able to code up a program of the form:

```
1:      Loop
2:      Exec    TexFetch
3:              TexFetch
4:              ALU
5:              ALU
6:              TexFetch
7:      End Loop
8:      ALU Export
```

But realize that 3: may be dependent on 2: and 4: is almost certainly dependent on 2: and 3:. Without clausing, these dependencies need to be expressed in the Control Flow instructions. Additionally, without separate 'texture clauses' and 'ALU clauses' we need to know which instructions to dispatch to the Texture Unit and which to the ALU unit. This information will be encapsulated in the flow control instructions.

Each control flow instruction will contain 2 bits of information for each (non-control flow) instruction:
a) ALU or Texture
b) Serialize Execution

(b) would force the thread to stop execution at this point (before the instruction is executed) and wait until all textures have been fetched. Given the allocation of reserved bits, this would mean that the count of an 'Exec' instruction would be limited to about 8 (non-control-flow) instructions. If more than this were needed, a second Exec (with the same conditions) would be issued.

Another function that relies upon 'clauses' is allocation and order of execution. We need to assure that pixels and vertices are exported in the correct order (even if not all execution is ordered) and that space in the output buffers are allocated in order. Additionally data can't be exported until space is allocated. A new control flow instruction:

Alloc <buffer select -- position,parameter, pixel or vertex memory. And the size required>.

would be created to mark where such allocation needs to be done. To assure allocation is done in order, the actual allocation for a given thread can not be performed unless the equivalent allocation for all previous threads is already completed. The implementation would also assure that execution of instruction(s) following the serialization due to the Alloc will occur in order -- at least until the next serialization or change from ALU to Texture. In most cases this will allow the exports to occur without any further synchronization. Only 'final' allocations or position allocations are

guaranteed to be ordered. Because strict ordering is required for pixels, parameters and positions, this implies only a single alloc for these structures. Vertex exports to memory do not require ordering during allocation and so multiple 'allocs' may be done.

6.2.1 *Control flow instructions table*

Here is the revised control flow instruction set.

Note that whenever a field is marked as RESERVED, it is assumed that all the bits of the field are cleared (0).

NOP		
47 … 44	43	42 … 0
0000	Addressing	RESERVED

This is a regular NOP.

Execute					
47 … 44	43	40 … 34	33 …16	15…12	11 … 0
0001	Addressing	RESERVED	Instructions type + serialize (9 instructions)	Count	Exec Address

Execute_End					
47 … 44	43	40 … 34	33 …16	15…12	11 … 0
0010	Addressing	RESERVED	Instructions type + serialize (9 instructions)	Count	Exec Address

Execute up to 9 instructions at the specified address in the instruction memory. The Instruction type field tells the sequencer the type of the instruction (LSB) (1 = Texture, 0 = ALU and whether to serialize or not the execution (MSB) (1 = Serialize, 0 = Non-Serialized). If Execute_End this is the last execution block of the shader program.

Conditional_Execute						
47 … 44	43	42	41 … 34	33…16	15 …12	11 … 0
0011	Addressing	Condition	Boolean address	Instructions type + serialize (9 instructions)	Count	Exec Address

Conditional_Execute_End						
47 … 44	43	42	41 … 34	33…16	15 …12	11 … 0
0100	Addressing	Condition	Boolean address	Instructions type + serialize (9 instructions)	Count	Exec Address

If the specified Boolean (8 bits can address 256 Booleans) meets the specified condition then execute the specified instructions (up to 9 instructions). If the condition is not met, we go on to the next control flow instruction. If Conditional_Execute_End and the condition is met, this is the last execution block of the shader program.

Conditional_Execute_Predicates							
47 … 44	43	42	41 … 36	35 … 34	33…16	15…12	11 … 0
0101	Addressing	Condition	RESERVED	Predicate vector	Instructions type + serialize (9 instructions)	Count	Exec Address

Conditional_Execute_Predicates_End							
47 … 44	43	42	41 … 36	35 … 34	33…16	15…12	11 … 0
0110	Addressing	Condition	RESERVED	Predicate vector	Instructions type + serialize (9 instructions)	Count	Exec Address

Check the AND/OR of all current predicate bits. If AND/OR matches the condition execute the specified number of instructions. We need to AND/OR this with the kill mask in order not to consider the pixels that aren't valid. If the

condition is not met, we go on to the next control flow instruction. If Conditional_Execute_Predicates_End and the condition is met, this is the last execution block of the shader program.

Conditional_Execute_Predicates_No_Stall							
47 … 44	43	42	41 … 36	35 … 34	33…16	15…12	11 … 0
1101	Addressing	Condition	RESERVED	Predicate vector	Instructions type + serialize (9 instructions)	Count	Exec Address

Conditional_Execute_Predicates_No_Stall_End							
47 … 44	43	42	41 … 36	35 … 34	33…16	15…12	11 … 0
1110	Addressing	Condition	RESERVED	Predicate vector	Instructions type + serialize (9 instructions)	Count	Exec Address

Same as Conditionnal_Execute_Predicates but the SQ is not going to wait for the predicate vector to be updated. You can only set this in the compiler if you know that the predicate set is only a refinement of the current one (like a nested if) because the optimization would still work.

Loop_Start					
47 … 44	43	42 … ~~17~~21	20 … 16	15…12	11 … 0
0111	Addressing	RESERVED	loop ID	RESERVED	Jump address

Loop Start. Compares the loop iterator with the end value. If loop condition not met jump to the address. Forward jump only. Also computes the index value. The loop id must match between the start to end, and also indicates which control flow constants should be used with the loop.

Loop_End						
47 … 44	43	42 … 24	23… 21	20 … 16	15…12	11 … 0
1000	Addressing	RESERVED	Predicate break	loop ID	RESERVED	start address

Loop end. Increments the counter by one, compares the loop count with the end value. If loop condition met, continue, else, jump BACK to the start of the loop. If predicate break != 0, then compares predicate vector n (specified by predicate break number). If all bits cleared then break the loop.

The way this is described does not prevent nested loops, and the inclusion of the loop id make this easy to do.

Conditionnal_Call						
47 … 44	43	42	41 … 34	33 … 13	12	11 … 0
1001	Addressing	Condition	Boolean address	RESERVED	Force Call	Jump address

If the condition is met, jumps to the specified address and pushes the control flow program counter on the stack. If force call is set the condition is ignored and the call is made always.

Return		
47 … 44	43	42 … 0
1010	Addressing	RESERVED

Pops the topmost address from the stack and jumps to that address. If nothing is on the stack, the program will just continue to the next instruction.

Conditionnal_Jump							
47 … 44	43	42	41… 34	33	32 … 13	12	11 … 0
1011	Addressing	Condition	Boolean address	FW only	RESERVED	Force Jump	Jump address

If force jump is set the condition is ignored and the jump is made always. If FW only is set then only forward jumps are allowed.

Allocate				
47 … 44	43	42…41	40 … 4	3 …0
1100	Debug	Buffer Select	RESERVED	Allocation size

Buffer Select takes a value of the following:
01 – position export (ordered export)
10 – parameter cache or pixel export (ordered export)
11 – pass thru (out of order exports).

Buffer Size takes a value of the following:
00 – 1 buffer
01 – 2 buffers
…
15 – 16 buffers

If debug is set this is a debug alloc (ignore if debug DB_ON register is set to off).

6.3 Implementation

The envisioned implementation has a buffer that maintains the state of each thread. A thread lives in a given location in the buffer during its entire life, but the buffer has FIFO qualities in that threads leave in the order that they enter. Actually two buffers are maintained -- one for Vertices and one for Pixels. The intended implementation would allow for:

16 entries for vertices
48 entries for pixels.

From each buffer, arbitration logic attempts to select 1 thread for the texture unit and 1 (interleaved) thread for the ALU unit. Once a thread is selected it is read out of the buffer, marked as invalid, and submitted to appropriate execution unit. It is returned to the buffer (at the same place) with its status updated once all possible sequential instructions have been executed. A switch from ALU to TEX or visa-versa or a Serialize_Execution modifier forces the thread to be returned to the buffer.

Each entry in the buffer will be stored across two physical pieces of memory - most bits will be stored in a 1 read port device. Only bits needed for thread arbitration will be stored in a highly multi-ported structure. The bits kept in the 1 read port device will be termed 'state'. The bits kept in the multi-read ported device will be termed 'status'.

'State Bits' needed include:

1. Control Flow Instruction Pointer (13 bits),
2. Execution Count Marker 4 bits),
3. Loop Iterators (4x9 bits),
4. Call return pointers (4x12 bits),
5. Predicate Bits (64 bits),
6. Export ID (1 bit),
7. Parameter Cache base Ptr (7 bits),
8. GPR Base Ptr (8 bits),
9. Context Ptr (3 bits).
10. LOD corrections (6x16 bits)
11. Valid bits (64 bits)

Absent from this list are 'Index' pointers. These are costly enough that I'm presuming that they are instead stored in the GPRs. The first seven fields above (Control Flow Ptr, Execution Count, Loop Counts, call return ptrs, Predicate bits, PC base ptr and export ID) are updated every time the thread is returned to the buffer based on how much progress has been mode on thread execution. GPR Base Ptr, Context Ptr and LOD corrections are unchanged throughout execution of the thread.

'Status Bits' needed include:

- Valid Thread
- Texture/ALU engine needed
- Texture Reads are outstanding
- Waiting on Texture Read to Complete
- Allocation Wait (2 bits)
- 00 – No allocation needed
- 01 – Position export allocation needed (ordered export)
- 10 – Parameter or pixel export needed (ordered export)
- 11 – pass thru (out of order export)
- Allocation Size (4 bits)
- Position Allocated
- First thread of a new context
- Event thread (NULL thread that needs to trickle down the pipe)
- Last (1 bit)
- Pulse SX (1 bit)

All of the above fields from all of the entries go into the arbitration circuitry. The arbitration circuitry will select a winner for both the Texture Engine and for the ALU engine. There are actually two sets of arbitration -- one for pixels and one for vertices. A final selection is then done between the two. But the rest of this implementation summary only considers the 'first' level selection which is similar for both pixels and vertices.

Texture arbitration requires no allocation or ordering so it is purely based on selecting the 'oldest' thread that requires the Texture Engine.

ALU arbitration is a little more complicated. First, only threads where either of Texture_Reads_outstanding or Waiting_on_Texture_Read_to_Complete are '0' are considered. Then if Allocation_Wait is active, these threads are further filtered based on whether space is available. If the allocation is position allocation, then the thread is only considered if all 'older' threads have already done their position allocation (position allocated bits set). If the allocation is parameter or pixel allocation, then the thread is only considered if it is the oldest thread. Also a thread is not considered if it is a parameter or pixel or position allocation, has its First_thread_of_a_new_context bit set and would cause ALU interleaving with another thread performing the same parameter or pixel or position allocation. Finally the 'oldest' of the threads that pass through the above filters is selected. If the thread needed to allocate, then at this time the allocation is done, based on Allocation_Size. If a thread has its “last” bit set, then it is also removed from the buffer, never to return.

If I now redefine 'clauses' to mean 'how many times the thread is removed from the thread buffer for the purpose of exection by either the ALU or Texture engine', then the minimum number of clauses needed is 2 -- one to perform the allocation for exports (execution automatically halts after an 'Alloc' instruction) (but doesn't performs the actual allocation) and one for the actual ALU/export instructions. As the 'Alloc' instruction could be part of a texture clause (presumably the final instruction in such a clause), a thread could still execute in this minimal number of 2 clauses, even if it involved texture fetching.

The Texture_Reads_Outstanding bit must be updated by the sequencer, based on keeping track of how many Texture Clauses have been executed by a given thread that have not yet had there data returned. Any number above 0 results in this bit being set. We could consider forcing synchronization such that two texture clauses for a given thread may not be outstanding at any time (that would be my preference for simplicity reasons and because it would require only very little change in the texture pipe interface). This would allow the sequencer to set the bit on execution of the texture clause, and allow the texture unit to return a pointer to the thread buffer on completion that clears the bit.

6.4 Data dependant predicate instructions

Data dependant conditionals will be supported in the R400. The only way we plan to support those is by supporting three vector/scalar predicate operations of the form:

PRED_SETE_# - similar to SETE except that the result is 'exported' to the sequencer.
PRED_SETNE_# - similar to SETNE except that the result is 'exported' to the sequencer.
PRED_SETGT_# - similar to SETGT except that the result is 'exported' to the sequencer
PRED_SETGTE_# - similar to SETGTE except that the result is 'exported' to the sequencer

For the scalar operations only we will also support the two following instructions:

PRED_SETE0_# – SETE0
PRED_SETE1_# – SETE1

The export is a single bit - 1 or 0 that is sent using the same data path as the MOVA instruction.   The sequencer will maintain 4 sets of 64 bit predicate vectors (in fact 8 sets because we interleave two programs but only 4 will be exposed) and use it to control the write masking. This predicate is not maintained across clause boundaries. The # sign is used to specify which predicate set you want to use 0 thru 3.

Then we have two conditional execute bits. The first bit is a conditional execute “on” bit and the second bit tells us if we execute on 1 or 0. For example, the instruction:

P0_ADD_# R0,R1,R2

Is only going to write the result of the ADD into those GPRs whose predicate bit is 0. Alternatively, P1_ADD_# would only write the results to the GPRs whose predicate bit is set. The use of the P0 or P1 without precharging the sequencer with a PRED instruction is undefined.

{Issue: do we have to have a NOP between PRED and the first instruction that uses a predicate?}

6.5 HW Detection of PV,PS

Because of the control program, the compiler cannot detect statically dependant instructions. In the case of non-masked writes and subsequent reads the sequencer will insert uses of PV,PS as needed. This will be done by comparing the read address and the write address of consecutive instructions. For masked writes, the sequencer will insert NOPs wherever there is a dependant read/write.

The sequencer will also have to insert NOPs between PRED_SET and MOVA instructions and their uses.

6.6 Register file indexing

Because we can have loops in fetch clause, we need to be able to index into the register file in order to retrieve the data created in a fetch clause loop and use it into an ALU clause. The instruction will include the base address for register indexing and the instruction will contain these controls:

Bit7	Bit 6	
0	0	'absolute register'
0	1	'relative register'
1	0	'previous vector'
1	1	'previous scalar'

In the case of an absolute register we just take the address as is. In the case of a relative register read we take the base address and we add to it the loop_index and this becomes our new address that we give to the shader pipe.

The sequencer is going to keep a loop index computed as such:

Index = Loop_iterator*Loop_step + Loop_start.

We loop until loop_iterator = loop_count. Loop_step is a signed value [-128…127]. The computed index value is a 10 bit counter that is also signed. Its real range is [-256,256]. The tenth bit is only there so that we can provide an out of range value to the “indexing logic” so that it knows when the provided index is out of range and thus can make the necessary arrangements.

6.7 Debugging the Shaders

In order to be able to debug the pixel/vertex shaders efficiently, we provide 2 methods.

6.7.1 *Method 1: Debugging registers*

Current plans are to expose 2 debugging, or error notification, registers:
1. address register where the first error occurred
2. count of the number of errors

The sequencer will detect the following groups of errors:
- count overflow
- constant indexing overflow
- register indexing overflow

Compiler recognizable errors:
- jump errors
    relative jump address > size of the control flow program
- call stack
    call with stack full
    return with stack empty

A jump error will always cause the program to break. In this case, a break means that a clause will halt execution, but allowing further clauses to be executed.

With all the other errors, program can continue to run, potentially to worst-case limits. The program will only break if the DB_PROB_BREAK register is set.

If indexing outside of the constant or the register range, causing an overflow error, the hardware is specified to return the value with an index of 0. This could be exploited to generate error tokens, by reserving and initializing the 0th register (or constant) for errors.

{ISSUE : Interrupt to the driver or not?}

6.7.2 *Method 2: Exporting the values in the GPRs*

1) The sequencer will have a debug active, count register and an address register for this mode.

Under the normal mode execution follows the normal course.

Under the debug mode it is assumed that the program is always exporting n debug vectors and that all other exports to the SX block (position, color, z, ect) will been turned off (changed into NOPs) by the sequencer (even if they occur before the address stated by the ADDR debug register).

7. Pixel Kill Mask

A vector of 64 bits is kept by the sequencer per group of pixels/vertices. Its purpose is to optimize the texture fetch requests and allow the shader pipe to kill pixels using the following instructions:

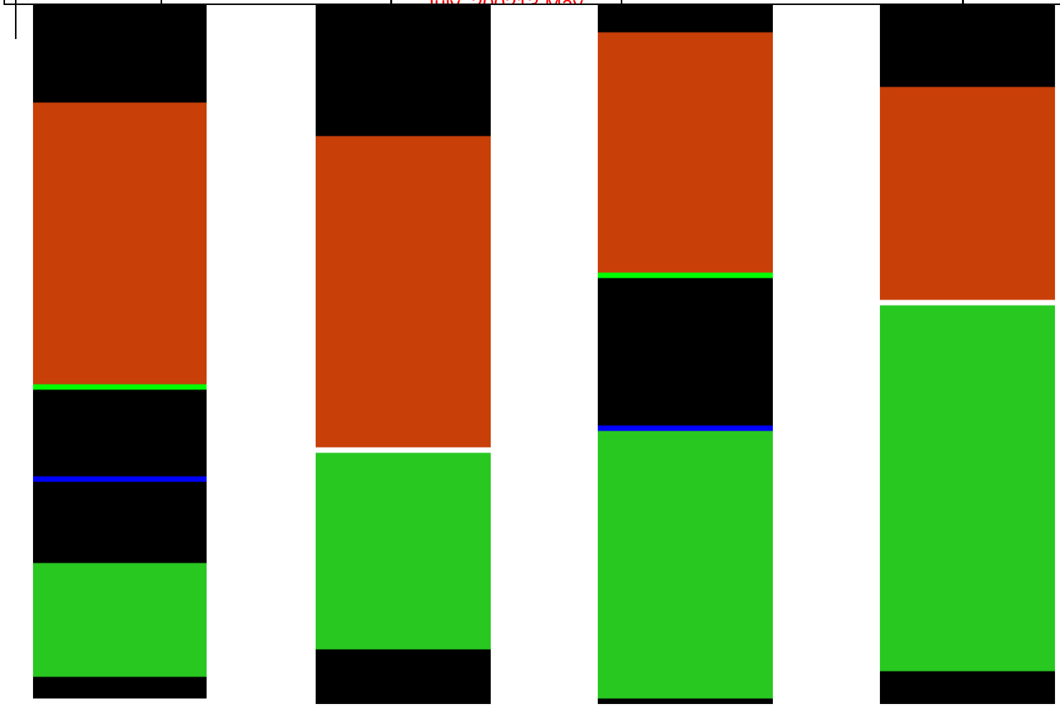
MASK_SETE
MASK_SETNE
MASK_SETGT
MASK_SETGTE

8. Multipass vertex shaders (HOS)

Multipass vertex shaders are able to export from the 6 last clauses but to memory ONLY.

9. Register file allocation

The register file allocation for vertices and pixels can either be static or dynamic. In both cases, the register file in managed using two round robins (one for pixels and one for vertices). In the dynamic case the boundary between pixels and vertices is allowed to move, in the static case it is fixed to 128-VERTEX_REG_SIZE for vertices and PIXEL_REG_SIZE for pixels.



Above is an example of how the algorithm works. Vertices come in from top to bottom; pixels come in from bottom to top. Vertices are in orange and pixels in green. The blue line is the tail of the vertices and the green line is the tail of the pixels. Thus anything between the two lines is shared. When pixels meets vertices the line turns white and the boundary is static until both vertices and pixels share the same “unallocated bubble”. Then the boundary is allowed to move again. The numbering of the GPRs starts from the bottom of the picture at index 0 and goes up to the top at index 127.

10. Fetch Arbitration

The fetch arbitration logic chooses one of the ~~8~~ n potentially pending fetch clauses to be executed. The choice is made by looking at the ~~fifos from 7 to 0~~Vs and Ps reservation stations and picking the first one ready to execute. Once chosen, the clause state machine will send one 2x2 fetch per clock (or 4 fetches in one clock every 4 clocks) until all the fetch instructions of the clause are sent. This means that there cannot be any dependencies between two fetches of the same clause.

The arbitrator will not wait for the fetches to return prior to selecting another clause for execution. The fetch pipe will be able to handle up to X(?) in flight fetches and thus there can be a fair number of active clauses waiting for their fetch return data.

11. ALU Arbitration

ALU arbitration proceeds in almost the same way than fetch arbitration. The ALU arbitration logic chooses one of the ~~8~~ n potentially pending ALU clauses to be executed. The choice is made by looking at the Vs and Ps reservation stations and picking the first one ready to execute~~The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute~~. There are two ALU arbiters, one for the even clocks and one for the odd clocks. For example, here is the sequencing of two interleaved ALU clauses (E and O stands for Even and Odd sets of 4 clocks):

Einst0 Oinst0 Einst1 Oinst1 Einst2 Oinst2 Einst0 Oinst3 Einst1 Oinst4 Einst2 Oinst0…

Proceeding this way hides the latency of 8 clocks of the ALUs. Also note that the interleaving also occurs across clause boundaries.

12. Handling Stalls

When the output file is full, the sequencer prevents the ALU arbitration logic from selecting the last clause (this way nothing can exit the shader pipe until there is place in the output file. If the packet is a vertex packet and the position buffer is full (POS_FULL) then the sequencer also prevents a thread from entering ~~the~~ an exporting clause. ~~(3?).~~ The sequencer will set the OUT_FILE_FULL signal n clocks before the output file is actually full and thus the ALU arbiter will be able read this signal and act accordingly by not preventing exporting clauses to proceed.

13. Content of the reservation station FIFOs

The reservation FIFOs contain the state of the vector of pixels and vertices. We have two sets of those: one for pixels, and one for vertices. They contain 3 bits of Render State 7 bits for the base address of the GPRs, some bits for LOD correction and coverage mask information in order to fetch fetch for only valid pixels, the quad address.

14. The Output File

The output file is where pixels are put before they go to the RBs. The write BW to this store is 256 bits/clock. Just before this output file are staging registers with write BW 512 bits/clock and read BW 256 bits/clock. The staging registers are 4x128 (and there are 16 of those on the whole chip).

15. IJ Format

The IJ information sent by the PA is of this format on a per quad basis:

We have a vector of IJ's (one IJ per pixel at the centroid of the fragment or at the center of the pixel depending on the mode bit). ~~The interpolation is done at a different precision across the 2x2. The upper left~~All pixel's parameters are always interpolated at full 20x24 mantissa precision. ~~Then the result of the interpolation along with the difference in IJ in reduced precision is used to interpolate the parameter for the other three pixels of the 2x2. Here is how we do it:~~

~~Assuming P0 is the interpolated parameter at Pixel 0 having the barycentric coordinates I(0), J(0) and so on for P1,P2 and P3. Also assuming that A is the parameter value at V0 (interpolated with I), B is the parameter value at V1 (interpolated with J) and C is the parameter value at V2 (interpolated with (1-I-J).~~

$$P0 = A + I(0)*(B - A) + J(0)*(C - A)$$
$$P1 = A + I(1)*(B - A) + J(1)*(C - A)$$
$$P2 = A + I(2)*(B - A) + J(2)*(C - A)$$
$$P3 = A + I(3)*(B - A) + J(3)*(C - A)$$

P0	P1
P2	P3

~~P0 is computed at 20x24 mantissa precision and P1 to P3 are computed at 8X24 mantissa precision. So far no visual degradation of the image was seen using this scheme.~~

Multiplies (Full Precision): ~~2~~8
~~Multiplies (Reduced precision): 6~~
Subtracts 19x24 (Parameters): 2
Adds: 8

FORMAT OF P~~0~~'s IJ : Mantissa 20 Exp 4 for I + Sign

Mantissa 20 Exp 4 for J + Sign

~~FORMAT of Deltas (x3):Mantissa 8 Exp 4 for I + Sign~~
~~Mantissa 8 Exp 4 for J + Sign~~

Total number of bits : 20*~~2~~ 8 ~~+ 8*6~~ + 4*8 + 4*2 = 200.
~~128~~

All numbers are kept using the un-normalized floating point convention: if exponent is different than 0 the number is normalized if not, then the number is un-normalized. The maximum range for the IJs (Full precision) is +/- ~~63~~ 1024.~~and the range for the Deltas is +/- 127.~~

15.1 Interpolation of constant attributes

Because of the floating point imprecision, we need to take special provisions if all the interpolated terms are the same or if two of the ~~barycentric coordinates~~terms are the same.

~~We start with the premise that if A = B and B = C and C = A, then P0,1,2,3 = A. Since one or more of the IJ terms may be zero, so we extend this to:~~

```
~~if (A=B and B=C and C=A)~~
~~  P0,1,2,3 = A;~~
~~else if ((I = 0) or (J = 0)) and~~
~~        ((J = 0) or (1-I-J = 0)) and~~
~~        ((1-J-I = 0) or (I = 0))) {~~
~~            if(I != 0) {~~
~~               P0 = A;~~
~~            } else if(J != 0) {~~
~~               P0 = B;~~
~~            } else {~~
~~               P0 = C;~~
~~            }~~
~~          //rest of the quad interpolated normally~~
~~}~~
~~else~~
~~{~~
~~          normal interpolation~~
~~}~~
```

16. Staging Registers

In order for the reuse of the vertices to be 14, the sequencer will have to re-order the data sent IN ORDER by the VGT for it to be aligned with the parameter cache memory arrangement. Given the following group of vertices sent by the VGT:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 || 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 || 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 || 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63

The sequencer will re-arrange them in this fashion:

0 1 2 3 16 17 18 19 32 33 34 35 48 49 50 51 || 4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55 || 8 9 10 11 24 25 26 27 40 41 42 43 56 57 58 59 || 12 13 14 15 28 29 30 31 44 45 46 47 60 61 62 63

The || markers show the SP divisions. In the event a shader pipe is broken, the VGT will send padding to account for the missing pipe. For example, if SP1 is broken, vertices 4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55 will still be sent by the VGT to the SQ **BUT** will not be processed by the SP and thus should be considered invalid (by the SU and VGT).

The most straightforward, *non-compressed* interface method would be to convert, in the VGT, the data to 32-bit floating point prior to transmission to the VSISRs. In this scenario, the data would be transmitted to (and stored in) the VSISRs in full 32-bit floating point. This method requires three 24-bit fixed-to-float converters in the VGT. Unfortunately, it also requires and additional 3,072 bits of storage across the VSISRs. This interface is illustrated in Figure 11~~Figure 11~~~~Figure 11~~. The area of the fixed-to-float converters and the VSISRs for this method is roughly estimated as 0.759sqmm using the R300 process. The gate count estimate is shown in Figure 10~~Figure 10~~~~Figure 10~~.

Basis for 8-deep Latch Memory (from R300)

8x24-bit	11631 μ^2	60.57813 μ^2 per bit
Area of 96x8-deep Latch Memory	46524 μ^2	
Area of 24-bit Fix-to-float Converter	4712 μ^2 per converter	

Method 1

Block	Quantity	Area
F2F	3	14136
8x96 Latch	16	744384
		758520 μ^2

Figure 10:Area Estimate for VGT to Shader Interface

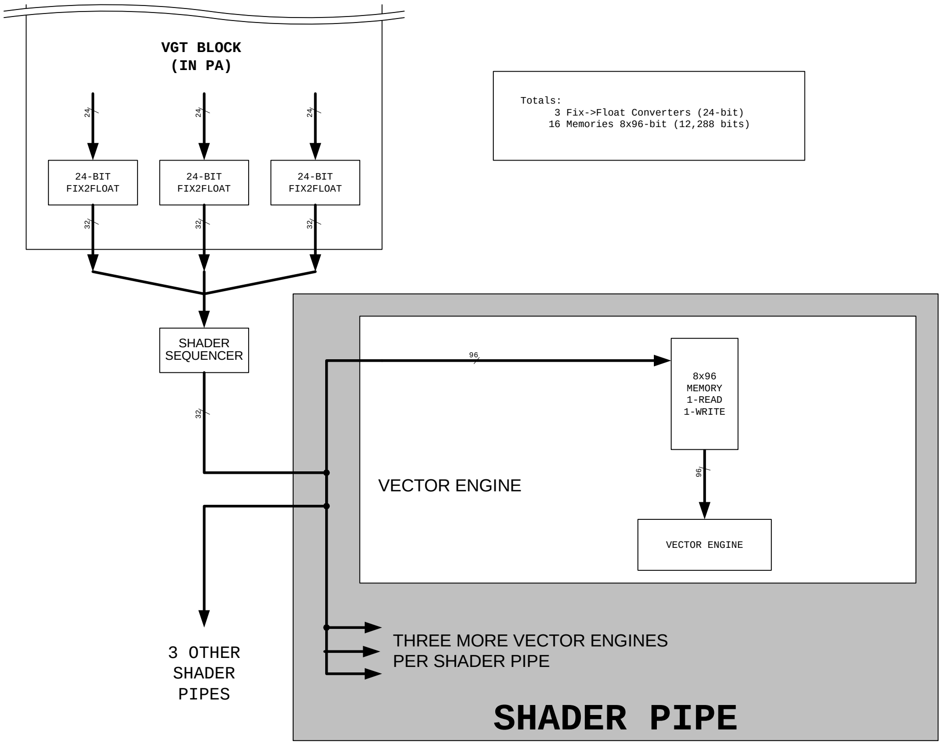


Figure 11:VGT to Shader Interface

17. The parameter cache

The parameter cache is where the vertex shaders export their data. It consists of 16 128x128 memories (1R/1W). The reuse engine will make it so that all vertexes of a given primitive will hit different memories. The allocation method for these memories is a simple round robin. The parameter cache pointers are mapped in the following way: 4MSBs are the memory number and the 7 LSBs are the address within this memory.

MEMORY NUMBER 4 bits	ADDRESS 7 bits

The PA generates the parameter cache addresses as the positions come from the SQ. All it needs to do is keep a Current_Location pointer (7 bits only) and as the positions comes increment the memory number. When the memory number field wraps around, the PA increments the Current_Location by VS_EXPORT_COUNT (a snooped register from the SQ). As an example, say the memories are all empty to begin with and the vertex shader is exporting 8 parameters per vertex (VS_EXPORT_COUNT = 8). The first position received is going to have the PC address 00000000000 the second one 00010000000, third one 00100000000 and so on up to 11110000000. Then the next position received (the 17th) is going to have the address 00000001000, the 18th 00010001000, the 19th 00100001000 and so on. The Current_location is NEVER reset BUT on chip resets. The only thing to be careful about is that if the SX doesn't send you a full group of positions (<64) then you need to fill the address space so that the next group starts correctly aligned (for example if you receive only 33 positions then you need to add 2*VS_EXPORT_COUNT to Current_Location and reset the memory count to 0 before the next vector begins).

17.1 Export restrictions

17.1.1 *Pixel exports:*

Pixels can export 1,2,3 or 4 color buffers to the SX(+z). The exports will be done in order. The PRED_OPTIMIZE function has to be turned of if the exports are done using interleaved predicated instructions. The exports will always be ordered to the SX.

17.1.2 *Vertex exports:*

Position or parameter caches can be exported in any order in the shader program. It is always better to export posistion as soon as possible. Position has to be exported in a single export block (no texture instructions can be placed between the exports). Parameter cache exports can be done in any order with texture instructions interleaved. The PRED_OPTIMIZE function has to be turned of if the exports are done using interleaved predicated instructions to the Parameter cache (see Arbitration restrictions for details). The exports will always be allocated in order to the SX.

17.1.3 *Pass thru exports:*

Pass thru exports have to be done in groups of the form:

```
Alloc 4 (8 or 12)
Execute ALU(ADDR) ALU(DATA) ALU(DATA) ALU(DATA)…
```

They cannot have texture instructions interleaved in the export block. These exports **are not guaranteed to be ordered**.

Also, when doing a pass thru export, Position MUST be exported AFTER all pass thru exports. This position export is used to synchronize the chip when doing a transition from pass thru shader to regular shader and vice versa.

17.2 Arbitration restrictions

Here are the Sequencer arbitration restrictions:

1) Cannot execute a serialized thread if the corresponding texture pending bit is set
2) Cannot allocate position if any older thread has not allocated position
3) If last thread is marked as not valid AND marked as last and we are about to execute the second to oldest thread also marked last then:

a. Both threads must be from the same context (cannot allow a first thread)
b. Must turn off the predicate optimization for the second thread

4) Cannot execute a texture clause if texture reads are pending
5) Cannot execute last if texture pending (even if not serial)

18. Export Types

The export type (or the location where the data should be put) is specified using the destination address field in the ALU instruction. Here is a list of all possible export modes:

18.1 Vertex Shading

0:15	- 16 parameter cache
16:31	- Empty (Reserved?)
32	- Export Address
33:40	- 8 vertex exports to the frame buffer and index
41:47	- Empty
48:55	- 8 debug export (interpret as normal vertex export)
60	- export addressing mode
61	- Empty
62	- position
63	- sprite size export that goes with position export (point_h,point_w,edgeflag,misc)

18.2 Pixel Shading

0	- Color for buffer 0 (primary)
1	- Color for buffer 1
2	- Color for buffer 2
3	- Color for buffer 3
4:7	- Empty
8	- Buffer 0 Color/Fog (primary)
9	- Buffer 1 Color/Fog
10	- Buffer 2 Color/Fog
11	- Buffer 3 Color/Fog
12:15	- Empty
16:31	- Empty (Reserved?)
32	- Export Address
33:40	- 8 exports for multipass pixel shaders.
41:47	- Empty
48:55	- 8 debug exports (interpret as normal pixel export)
60	- export addressing mode
61:62	- Empty
63	- Z for primary buffer (Z exported to 'alpha' component)

19. Special Interpolation modes

19.1 Real time commands

We are unable to use the parameter memory since there is no way for a command stream to write into it. Instead we need to add three 16x128 memories (one for each of three vertices x 16 interpolants). These will be mapped onto the register bus and written by type 0 packets, and output to the the parameter busses (the sequencer and/or PA need to be able to address the realtime parameter memory as well as the regular parameter store. For higher performance we should be able able to view them as two banks of 16 and do double buffering allowing one to be loaded, while the other is rasterized with. Most overlay shaders will need 2 or 4 scalar coordinates, one option might be to restrict the memory to 16x64 or 32x64 allowing only two interpolated scalars per cycle, the only problem I see with this is, if we view support for 16 vector-4 interpolants important (true only if we map Microsoft's high priority stream to the realtime stream), then the PA/sequencer need to support a realtime-specific mode where we need to address 32 vectors of parameters instead of 16. This mode is triggered by the primitive type: REAL TIME. The actual memories are in the in the SX blocks. The parameter data memories are hooked on the RBBM bus and are loaded by the CP using register mapped memory.

19.2 Sprites/ XY screen coordinates/ FB information

When working with sprites, one may want to overwrite the parameter 0 with SC generated data. Also, XY screen coordinates may be needed in the shader program. This functionality is controlled by the gen_I0 register (in SQ) in conjunction with the SND_XY register (in SC). Also it is possible to send the faceness information (for OGL front/back special operations) to the shader using the same control register. Here is a list of all the modes and how they interact together:

Gen_st is a bit taken from the interface between the SC and the SQ. This is the MSB of the primitive type. If the bit is set, it means we are dealing with Point AA, Line AA or sprite and in this case the vertex values are going to generated between 0 and 1.

Param_Gen_I0 disable, snd_xy disable, no gen_st – I0 = No modification
Param_Gen_I0 disable, snd_xy disable, gen_st – I0 = No modification
Param_Gen_I0 disable, snd_xy enable, no gen_st – I0 = No modification
Param_Gen_I0 disable, snd_xy enable, gen_st – I0 = No modification
Param_Gen_I0 enable, snd_xy disable, no gen_st – I0 = garbage, garbage, garbage, faceness
Param_Gen_I0 enable, snd_xy disable, gen_st – I0 = garbage, garbage, s, t
Param_Gen_I0 enable, snd_xy enable, no gen_st – I0 = screen x, screen y, garbage, faceness
Param_Gen_I0 enable, snd_xy enable, gen_st – I0 = screen x, screen y, s, t

19.3 Auto generated counters

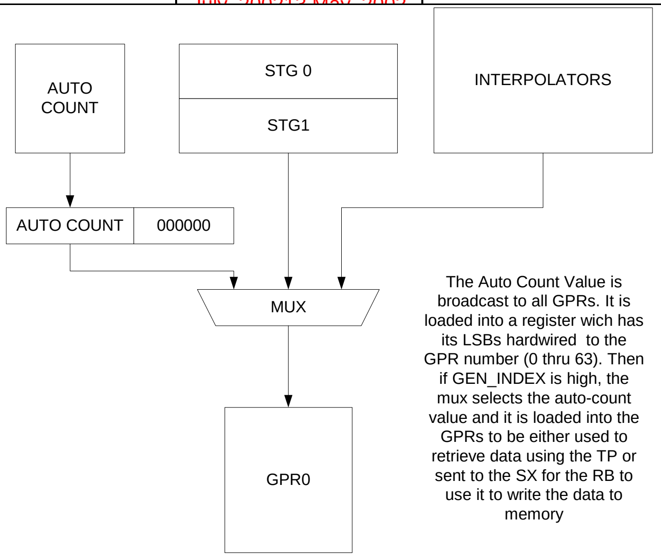
In the cases we are dealing with multipass shaders, the sequencer is going to generate a vector count to be able to both use this count to write the 1st pass data to memory and then use the count to retrieve the data on the 2nd pass. The count is always generated in the same way but it is passed to the shader in a slightly different way depending on the shader type (pixel or vertex). This is toggled on and off using the GEN_INDEX register. The sequencer is going to keep two counters, one for pixels and one for vertices. Every time a full vector of vertices or pixels is written to the GPRs the counter is incremented. Every time a state change is detected, the corresponding counter is reset. While there is only one count broadcast to the GPRs, the LSB are hardwired to specific values making the index different for all elements in the vector.

19.3.1 *Vertex shaders*

In the case of vertex shaders, if GEN_INDEX is set, the data will be put into the x field of the third register (it means that the compiler must allocate 3 GPRs in all multipass vertex shader modes).

19.3.2 *Pixel shaders*

In the case of pixel shaders, if GEN_INDEX is set and Param_Gen_I0 is enabled, the data will be put in the x field of the 2nd register (R1.x), else if GEN_INDEX is set the data will be put into the x field of the 1st register (R0.x).



The Auto Count Value is broadcast to all GPRs. It is loaded into a register wich has its LSBs hardwired to the GPR number (0 thru 63). Then if GEN_INDEX is high, the mux selects the auto-count value and it is loaded into the GPRs to be either used to retrieve data using the TP or sent to the SX for the RB to use it to write the data to memory

Figure 12: GPR input mux Control

20. State management

Every clock, the sequencer will report to the CP the oldest states still in the pipe. These are the states of the programs as they enter the last ALU clause.

20.1 Parameter cache synchronization

In order for the sequencer not to begin a group of pixels before the associated group of vertices has finished, the sequencer will keep a 6 bit count per state (for a total of 8 counters). These counters are initialized to 0 and every time a vertex shader exports its data TO THE PARAMETER CACHE, the corresponding pointer is incremented. When the SC sends a new vector of pixels with the SC_SQ_new_vector bit asserted, the sequencer will first check if the count is greater than 0 before accepting the transmission (it will in fact accept the transmission but then lower its ready to receive). Then the sequencer waits for the count to go to one and decrements it. The sequencer can then issue the group of pixels to the interpolators. Every time the state changes, the new state counter is initialized to 0.

21. XY Address imports

The SC will be able to send the XY addresses to the GPRs. It does so by interleaving the writes of the IJs (to the IJ buffer) with XY writes (to the XY buffer). Then when writing the data to the GPRs, the sequencer is going to interpolate the IJ data or pass the XY data thru a Fix→float converter and expander and write the converted values to the GPRs. The Xys are currently SCREEN SPACE COORDINATES. The values in the XY buffers will wrap. See section 19.2 for details on how to control the interpolation in this mode.

21.1 Vertex indexes imports

In order to import vertex indexes, we have 16 8x96 staging registers. These are loaded one line at a time by the VGT block (96 bits). They are loaded in floating point format and can be transferred in 4 or 8 clocks to the GPRs.

22. Registers

Please see the auto-generated web pages for register definitions.~~Control~~

~~REG_DYNAMIC Dynamic allocation (pixel/vertex) of the register file on or off.~~

~~REG_SIZE_PIX Size of the register file's pixel portion (minimal size when dynamic allocation turned on)~~
~~REG_SIZE_VTX Size of the register file's vertex portion (minimal size when dynamic allocation turned on)~~
~~ARBITRATION_POLICY policy of the arbitration between vertexes and pixels~~
~~INST_BASE_VTX start point for the vertex instruction store (RT always ends at vertex_base and Begins at 0)~~
~~INST_BASE_PIX start point for the pixel shader instruction store~~
~~ONE_THREAD debug state register. Only allows one program at a time into the GPRs~~
~~ONE_ALU debug state register. Only allows one ALU program at a time to be executed (instead of 2)~~
~~INSTRUCTION This is where the CP puts the base address of the instruction writes and type (auto-incremented on reads/writes) Register mapped~~
~~CONSTANTS 512*4 ALU constants + 32*6 Texture state 32 bits registers (logically mapped)~~
~~CONSTANTS_RT 256*4 ALU constants + 32*6 texture states? (physically mapped)~~
~~CONSTANT_EO_RT This is the size of the space reserved for real time in the constant store (from 0 to CONSTANT_EO_RT). The re-mapping table operates on the rest of the memory~~
~~TSTATE_EO_RT This is the size of the space reserved for real time in the fetch state store (from 0 to TSTATE_EO_RT). The re-mapping table operates on the rest of the memory~~

22.2 ~~Context~~

Formatted: Bullets and Numbering

~~PS_BASE base pointer for the pixel shader in the instruction store~~
~~VS_BASE base pointer for the vertex shader in the instruction store~~
~~VS_CF_SIZE size of the vertex shader (# of instructions in control program/2)~~
~~PS_CF_SIZE size of the pixel shader (# of instructions in control program/2)~~
~~PS_SIZE size of the pixel shader (cntl+instructions)~~
~~VS_SIZE size of the vertex shader (cntl+instructions)~~
~~PS_NUM_REG number of GPRs to allocate for pixel shader programs~~
~~VS_NUM_REG number of GPRs to allocate for vertex shader programs~~
~~PARAM_SHADE One 16 bit register specifying which parameters are to be gouraud shaded (0 = flat, 1 = gouraud)~~
~~PARAM_WRAP 64 bits: for which parameters (and channels (xyzw)) do we do the cyl wrapping (0=linear, 1=cylindrical).~~
~~PS_EXPORT_MODE 0xxxx : Normal mode~~
~~1xxxx : Multipass mode~~
~~If normal, bbbz where bbb is how many colors (0-4) and z is export z or not~~
~~If multipass 1-12 exports for color.~~
~~VS_EXPORT_MODE 0: position (1 vector), 1: position (2 vectors), 3:multipass~~
~~VS_EXPORT_COUNT Number of locations exported by the VS (and thus number of interpolated parameters)~~
~~PARAM_GEN_I0 Do we overwrite or not the parameter 0 with XY data and generated T and S values~~
~~GEN_INDEX Auto generates an address from 0 to XX. Puts the results into R0-1 for pixel shaders and R2 for vertex shaders~~
~~CONST_BASE_VTX (9 bits)Logical Base address for the constants of the Vertex shader~~
~~CONST_BASE_PIX (9 bits) Logical Base address for the constants of the Pixel shader~~
~~CONST_SIZE_PIX (8 bits) Size of the logical constant store for pixel shaders~~
~~CONST_SIZE_VTX (8 bits) Size of the logical constant store for vertex shaders~~
~~INST_PRED_OPTIMIZE Turns on the predicate bit optimization (if of, conditional_execute_predicates is always executed).~~
~~CF_BOOLEANS 256 boolean bits~~
~~CF_LOOP_COUNT 32x8 bit counters (number of times we traverse the loop)~~
~~CF_LOOP_START 32x8 bit counters (init value used in index computation)~~
~~CF_LOOP_STEP 32x8 bit counters (step value used in index computation)~~

~~23. DEBUG Registers~~

Formatted: Bullets and Numbering

~~23.1 Context~~

~~DB_PROB_ADDR~~	~~instruction address where the first problem occurred~~
~~DB_PROB_COUNT~~	~~number of problems encountered during the execution of the program~~
~~DB_PROB_BREAK~~	~~break the clause if an error is found.~~
~~DB_ON~~	~~turns on an off debug method 2~~
~~DB_INST_COUNT~~	~~instruction counter for debug method 2~~
~~DB_BREAK_ADDR~~	~~break address for method number 2~~

~~23.2 Control~~

Formatted: Bullets and Numbering

~~DB_ALUCST_MEMSIZE~~	~~Size of the physical ALU constant memory~~
~~DB_TSTATE_MEMSIZE~~	~~Size of the physical texture state memory~~

~~24.~~23. Interfaces

Formatted: Bullets and Numbering

~~24.1~~23.1 External Interfaces

Whenever an x is used, it means that the bus is broadcast to all units of the same name. For example, if a bus is named SQ→SPx it means that SQ is going to broadcast the same information to all SP instances.

~~24.2~~23.2 SC to SP Interfaces

Formatted: Bullets and Numbering

~~24.2.1~~23.2.1 *SC_SP#*

Formatted: Bullets and Numbering

There is one of these interfaces at front of each of the SP (buffer to stage pixel interpolators). This interface transmits the I,J data for pixel interpolation. For the entire system, two quads per clock are transferred to the 4 SPs, so each of these 4 interfaces transmits one half of a quad per clock. The interface below describes a half of a quad worth of data.
The actual data which is transferred per quad is

Ref Pix I => S4.20 Floating Point I value *4
Ref Pix J => S4.20 Floating Point J value *4
~~Delta Pix I (x3) => S4.8 Floating Point Delta I value~~
~~Delta Pix J (x3) => S4.8 Floating Point Delta J value~~

This equates to a total of ~~128~~ 200 bits which transferred over 2 clocks
and therefor needs an interface ~~64~~100 bits wide

Additionally, X,Y data (12-bit unsigned fixed) is conditionally sent across this data bus over the same wires in an additional clock. The X,Y data is sent on the lower 24 bits of the data bus with faceness in the msb.
Transfers across these interfaces are synchronized with the SC_SQ IJ Control Bus transfers.

The data transfer across each of these busses is controlled by a IJ_BUF_INUSE_COUNT in the SC. Each time the SC has sent a pixel vector's worth of data to the SPs, he will increment the IJ_BUF_INUSE_COUNT count. Prior to sending the next pixel vectors data, he will check to make sure the count is less than MAX_BUFER_MINUS_2, if not the SC will stall until the SQ returns a pipelined pulse to decrement the count when he has scheduled a buffer free. Note: We could/may optimize for the case of only sending only IJ to use all the buffers to pre-load more. Currently it is planned for the SP to hold 2 double buffers of I,J data and two buffers of X,Y data, so if either X,Y or Centers and Centroids are on, then the SC can send two Buffers.

In at least the initial version, the SC shall send 16 quads per pixel vector even if the vector is not full. This will increment buffer write address pointers correctly all the time. (We may revisit this for both the SX,SP,SQ and add a EndOfVector signal on all interfaces to quit early. We opted for the simple mode first with a belief that only the end of packet and multiple new vector signals should cause a partial vector and that this would not really be significant performance hit.)

Name	Bits	Description
SC_SP#_data	~~64~~100	IJ information sent over 2 clocks (or X,Y in 24 LSBs with faceness in upper bit) **Type 0 or 1**, First clock I, second clk J Field ULC URC LLC LRC Bits [63:39] [38:26] [25:13] [12:0] Format SE4M20 - SE4M20~~SE4M8~~ –SE4M20~~SE4M8~~ -SE4M20~~SE4M8~~ **Type 2** Field Face X Y Bits [63] [23:12] [11:0] Format Bit Unsigned Unsigned
SC_SP#_valid	1	Valid
SC_SP#_last_quad_data	1	This bit will be set on the last transfer of data per quad.
SC_SP#_type	2	0 -> Indicates centroids 1 -> Indicates centers 2 -> Indicates X,Y Data and faceness on data bus The SC shall look at state data to determine how many types to send for the interpolation process.

The # is included for clarity in the spec and will be replaced with a prefix of u#_ in the verilog module statement for the SC and the SP block will have neither because the instantiation will insert the prefix.

~~24.2.2~~23.2.2 *SC_SQ*

Formatted: Bullets and Numbering

This is the control information sent to the sequencer in order to synchronize and control the interpolation and/or loading data into the GPRs needed to execute a shader program on the sent pixels. This data will be sent over two clocks per transfer with 1 to 16 transfers. Therefore the bus (approx 94 bits) could be folded in half to approx 49 bits.

Name	Bits	Description
SC_SQ_data	46	Control Data sent to the SQ 1 clk transfers Event – valid data consist of event_id and state_id. Instruct SQ to post an event vector to send state id and event_id through request fifo and onto the reservation stations making sure state id and/or event_id gets back to the CP. Events only follow end of packets so no pixel vectors will be in progress. Empty Quad Mask – Transfer Control data consisting of pc_dealloc or new_vector. Receipt of this is to transfer pc_dealloc or new_vector without any valid quad data. New vector will always be posted to request fifo and pc_dealloc will be attached to any pixel vector outstanding or posted in request fifo if no valid quad outstanding. 2 clk transfers Quad Data Valid – Sending quad data with or without new_vector or pc_dealloc. New vector will be posted to request fifo with or without a pixel vector and

		pc_dealloc will be posted with a pixel vector unless none is in progress. In this case the pc_dealloc will be posted in the request queue. Filler quads will be transferred with The Quad mask set but the pixel corresponding pixel mask set to zero.
SC_SQ_valid	1	SC sending valid data, 2nd clk could be all zeroes

SC_SQ_data – first clock and second clock transfers are shown in the table below.

Name	BitField	Bits	Description
1st Clock Transfer			
SC_SQ_event	0	1	This transfer is a 1 clock event vector Force quad_mask = new_vector=pc_dealloc=0
SC_SQ_event_id	[4:1]	4	This field identifies the event 0 => denotes an End Of State Event 1 => TBD
SC_SQ_pc_dealloc	[7:5]	3	Deallocation token for the Parameter Cache
SC_SQ_new_vector	8	1	The SQ must wait for Vertex shader done count > 0 and after dispatching the Pixel Vector the SQ will decrement the count.
SC_SQ_quad_mask	[12:9]	4	Quad Write mask left to right SP0 => SP3
SC_SQ_end_of_prim	13	1	End Of the primitive
SC_SQ_state_id	[16:14]	3	State/constant pointer (6*3+3)
SC_SQ_pix_mask	[32:17]	16	Valid bits for all pixels SP0=>SP3 (UL,UR,LL,LR)
SC_SQ_provok_vtx	[37:36]	2	Provoking vertex for flat shading
SC_SQ_pc_ptr0	[48:38]	11	Parameter Cache pointer for vertex 0
2nd Clock Transfer			
SC_SQ_pc_ptr1	[10:0]	11	Parameter Cache pointer for vertex 1
SC_SQ_pc_ptr2	[21:11]	11	Parameter Cache pointer for vertex 2
SC_SQ_lod_correct	[45:22]	24	LOD correction per quad (6 bits per quad)
SC_SQ_prim_type	[48:46]	3~~3~~	Stippled line and Real time command need to load tex cords from alternate buffer 0~~0~~00: ~~Normal~~ Sprite (point) 001: Line 010: Tri_rect 10~~10~~0: ~~Realtime~~ Realtime Sprite (point) 101: Realtime Line 110: Realtime Tri_rect~~101: Line AA 110: Point AA (Sprite)~~

Name	Bits	Description
SQ_SC_free_buff	1	Pipelined bit that instructs SC to decrement count of buffers in use.
SQ_SC_dec_cntr_cnt	1	Pipelined bit that instructs SC to decrement count of new vector and/or event sent to prevent SC from overflowing SQ interpolator/Reservation request fifo.

The scan converter will submit a partial vector whenever:

1.) He gets a primitive marked with an end of packet signal.
2.) A current pixel vector is being assembled with at least one or more valid quads and the vector has been marked for deallocate when a primitive marked new_vector arrives. The Scan Converter will submit a partial vector (up to 16quads with zero pixel mask to fill out the vector) prior to submitting the new_vector marker\primitive.

(This will prevent a hang which can be demonstrated when all primitives in a packet three vectors are culled except for a one quad primitive that gets marked pc_dealloc (vertices maximum size). In this case two new_vectors are submitted and processed, but then one valid quad with the pc_dealloc creates a vector and then the new would wait for another vertex vector to be processed, but the one being waited for could never export until the pc_dealloc signal made it through and thus the hang.)

~~24.2.3~~23.2.3 *SQ to SX(SP): Interpolator bus*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SP~~X~~x_interp_flat_vtx	SQ→SPx	2	Provoking vertex for flat shading
SQ_SP~~X~~x_interp_flat_gouraud	SQ→SPx	1	Flat or gouraud shading
SQ_~~SXx~~SPx_interp_cyl_wrap	SQ→SPx	4	Wich channel needs to be cylindrical wrapped
SQ_SXx_pc_ptr0	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_pc_ptr1	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_pc_ptr2	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_rt_sel	SQ→SXx	1	Selects between RT and Normal data
SQ_SXx_pc_wr_en	SQ→SXx	1	Write enable for the PC memories
SQ_SXx_pc_wr_addr	SQ→SXx	7	Write address for the PCs
SQ_SXx_pc_channel_mask	SQ→SXx	4	Channel mask
SQ_SXx_pc_ptr_valid	SQ→SXx	1	Read pointers are valid.
SQ_SPx_interp_valid	SQ→SPx	1	Interpolation control valid

~~24.2.4~~23.2.4 *SQ to SP: Staging Register Data*

Formatted: Bullets and Numbering

This is a broadcast bus that sends the VSISR information to the staging registers of the shader pipes.

Name	Direction	Bits	Description
SQ_SPx_vsr_data	SQ→SPx	96	Pointers of indexes or HOS surface information
SQ_SPx_vsr_double	SQ→SPx	1	0: Normal 96 bits per vert 1: double 192 bits per vert
SQ_SP0_ vsr_valid	SQ→SP0	1	Data is valid
SQ_SP1_ vsr_ valid	SQ→SP1	1	Data is valid
SQ_SP2_ vsr_ valid	SQ→SP2	1	Data is valid
SQ_SP3_ vsr_ valid	SQ→SP3	1	Data is valid
SQ_SPx_vsr_read	SQ→SPx	1	Increment the read pointers

~~24.2.5~~23.2.5 *VGT to SQ : Vertex interface*

Formatted: Bullets and Numbering

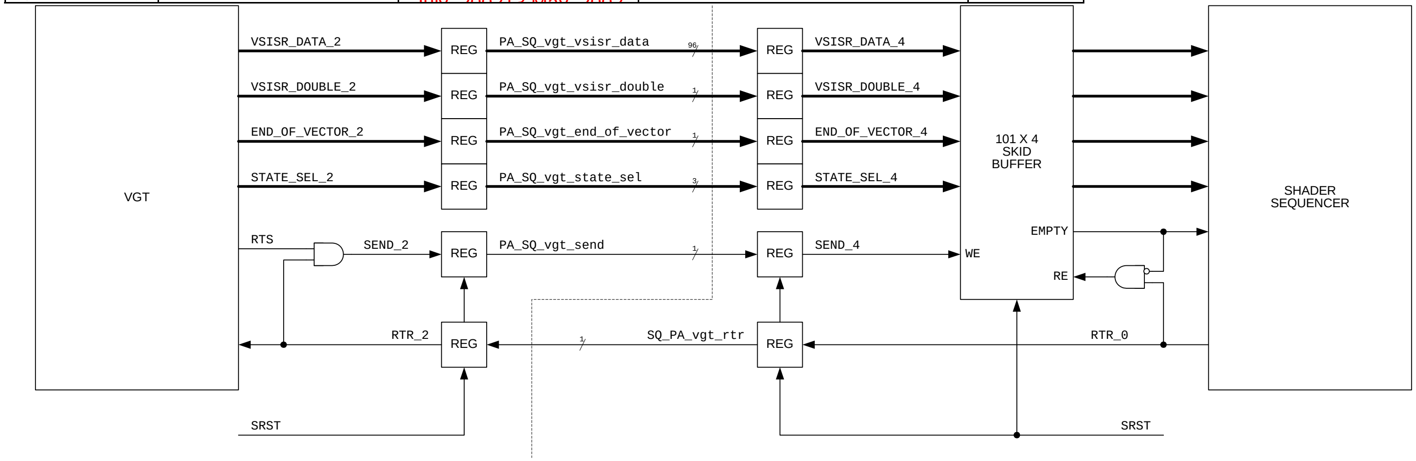
~~24.2.5.1~~23.2.5.1 Interface Signal Table

The area difference between the two methods is not sufficient to warrant complicating the interface or the state requirements of the VSISRs. **Therefore, the POR for this interface is that the VGT will transmit the data to the VSISRs (via the Shader Sequencer) in full, 32-bit floating-point format.** The VGT can transmit up to six 32-bit floating-point values to each VSISR where four or more values require two transmission clocks. The data bus is 96 bits wide.

Name	Bits	Description
VGT_SQ_vsisr_data	96	Pointers of indexes or HOS surface information
VGT_SQ_event	1	VGT is sending an event
VGT_SQ_vsisr_~~double~~continued	1	0: Normal 96 bits per vert 1: double 192 bits per vert
VGT_SQ_end_of_~~vector~~vtx_vect	1	Indicates the last VSISR data set for the current process vector (for double vector data, "end_of_vector" is set on the first vector)
VGT_SQ_indx_valid	1	Vsisr data is valid
VGT_SQ_state	3	Render State (6*3+3 for constants). This signal is guaranteed to be correct when “VGT_SQ_vgt_end_of_vector” is high.
VGT_SQ_send	1	Data on the VGT_SQ is valid receive (see write-up for standard R400 SEND/RTR interface handshaking)
SQ_VGT_rtr	1	Ready to receive (see write-up for standard R400 SEND/RTR interface handshaking)

~~24.2.5.2~~23.2.5.2 Interface Diagrams

Formatted: Bullets and Numbering



VGT
VSISR_DATA_2
VSISR_DOUBLE_2
END_OF_VECTOR_2
STATE_SEL_2
RTS
SEND_2
RTR_2
SRST
REG
REG
REG
REG
REG
REG
PA_SQ_vgt_vsisr_data
96
PA_SQ_vgt_vsisr_double
1
PA_SQ_vgt_end_of_vector
1
PA_SQ_vgt_state_sel
3
PA_SQ_vgt_send
1
1
SQ_PA_vgt_rtr
REG
REG
REG
REG
REG
REG
VSISR_DATA_4
VSISR_DOUBLE_4
END_OF_VECTOR_4
STATE_SEL_4
SEND_4
101 X 4
SKID
BUFFER
EMPTY
WE
RE
RTR_0
SRST
SHADER
SEQUENCER

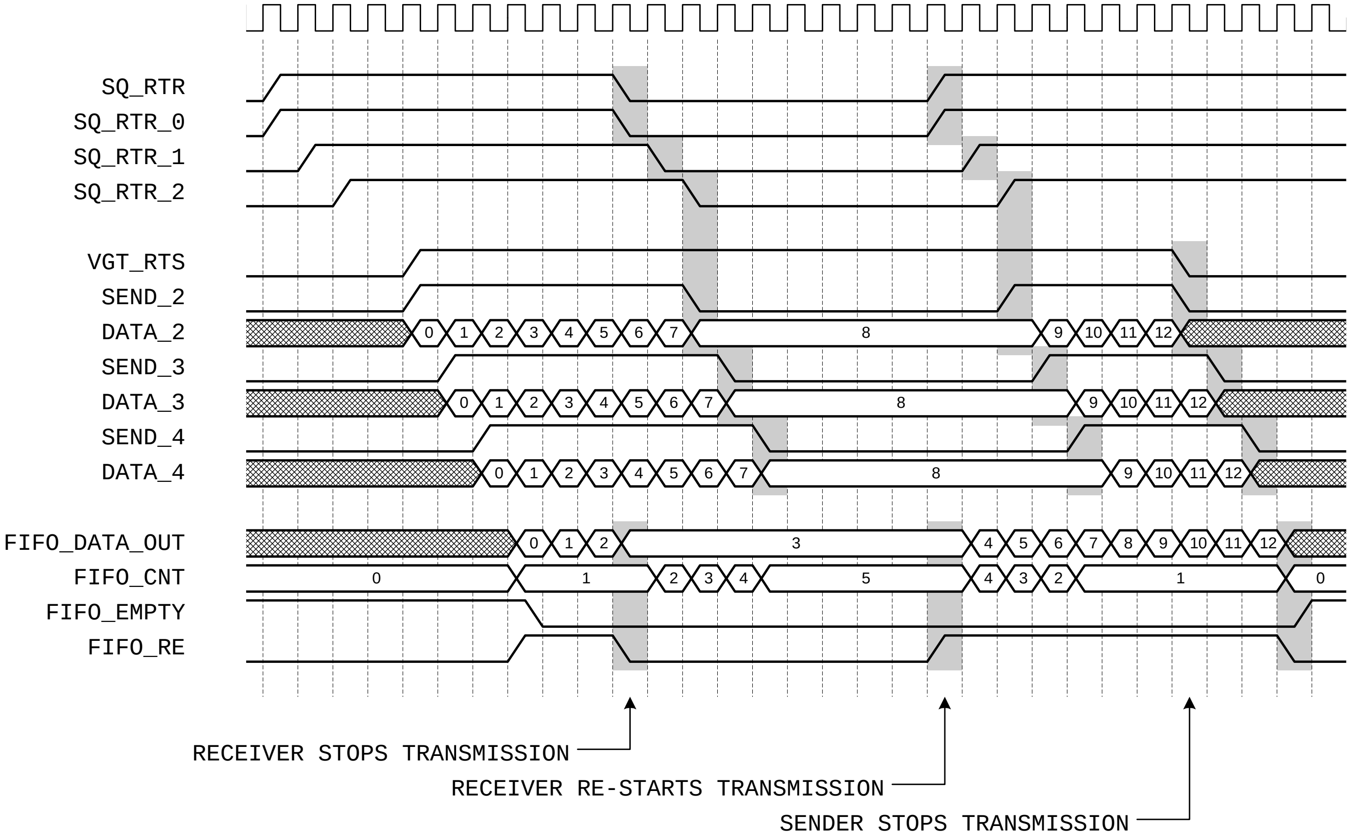


Figure 1. Detailed Logical Diagram for PA_SQ_vgt Interface.

~~24.2.6~~23.2.6 SQ to SX: Control bus

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SXx_exp_type	SQ→SXx	2	00: Pixel without z (1 to 4 buffers) 01: Pixel with z (1 to 4 buffers) 10: Position (1 or 2 results) 11: Pass thru (4,8 or 12 results aligned)
SQ_SXx_exp_number	SQ→SXx	2	Number of locations needed in the export buffer (encoding depends on the type see bellow).
SQ_SXx_exp_alu_id	SQ→SXx	1	ALU ID
SQ_SXx_exp_valid	SQ→SXx	1	Valid bit
SQ_SXx_exp_state	SQ→SXx	3	State Context
SQ_SXx_free_done	SQ→SXx	1	Pulse to indicate that the previous export is finished (this can be sent with or without the other fields of the interface)
SQ_SXx_free_alu_id	SQ→SXx	1	ALU ID

Depending on the type the number of export location changes:

- Type 00 : Pixels without Z
    - 00 = 1 buffer
    - 01 = 2 buffers
    - 10 = 3 buffers
    - 11 = 4 buffer
- Type 01: Pixels with Z
    - 00 = 2 Buffers (color + Z)
    - 01 = 3 buffers (2 color + Z)
    - 10 = 4 buffers (3 color + Z)
    - 11 = 5 buffers (4 color + Z)
- Type 10 : Position export
    - 00 = 1 position
    - 01 = 2 positions
    - 1X = Undefined
- Type 11: Pass Thru
    - 00 = 4 buffers
    - 01 = 8 buffers
    - 10 = 12 buffers
    - 11 = Undefined

Below the thick black line is the end of transfer packet that tells the SX that a given export is finished. The report packet **will always arrive either before or at the same time than the next export to the same ALU id**.

~~24.2.7~~23.2.7 SX to SQ : Output file control

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SXx_SQ_exp_count_rdy	SXx→SQ	1	Raised by SX0 to indicate that the following two fields reflect the result of the most recent export
SXx_SQ_exp_pos_avail	SXx→SQ	1	Specifies whether there is room for another position.
SXx_SQ_exp_buf_avail	SXx→SQ	7	Specifies the space available in the output buffers. 0: buffers are full 1: 2K-bits available (32-bits for each of the 64 pixels in a clause) ... 64: 128K-bits available (16 128-bit entries for each of 64 pixels) 65-127: RESERVED

~~24.2.8~~23.2.8 SQ to TP: Control bus

Formatted: Bullets and Numbering

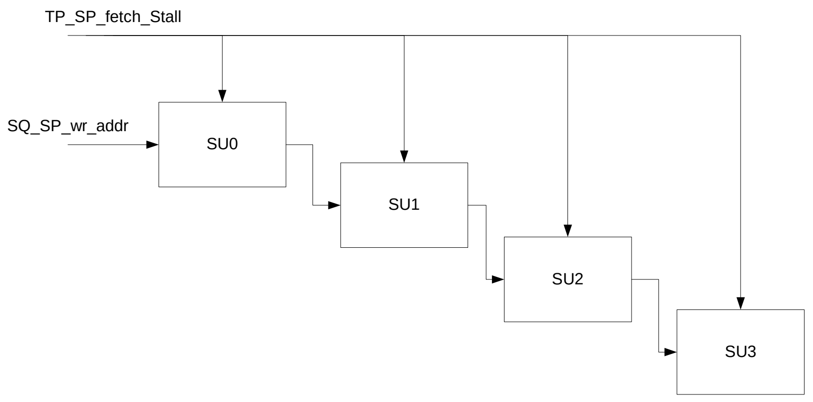
Once every clock, the fetch unit sends to the sequencer on which RS line it is now working and if the data in the GPRs is ready or not. This way the sequencer can update the fetch valid bits flags for the reservation station. The sequencer also provides the instruction and constants for the fetch to execute and the address in the register file where to write the fetch return data.

Name	Direction	Bits	Description
TPx_SQ_data_rdy	TPx→ SQ	1	Data ready
TPx_SQ_rs_line_num	TPx→ SQ	6	Line number in the Reservation station
TPx_SQ_type	TPx→ SQ	1	Type of data sent (0:PIXEL, 1:VERTEX)
SQ_TPx_send	SQ→TPx	1	Sending valid data
SQ_TPx_const	SQ→TPx	48	Fetch state sent over 4 clocks (192 bits total)
SQ_TPx_instr	SQ→TPx	24	Fetch instruction sent over 4 clocks
SQ_TPx_end_of_group	SQ→TPx	1	Last instruction of the group
SQ_TPx_Type	SQ→TPx	1	Type of data sent (0:PIXEL, 1:VERTEX)
SQ_TPx_gpr_phase	SQ→TPx	2	Write phase signal
SQ_TP0_lod_correct	SQ→TP0	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP0_pix_mask	SQ→TP0	4	Pixel mask 1 bit per pixel
SQ_TP1_lod_correct	SQ→TP1	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP1_pix_mask	SQ→TP1	4	Pixel mask 1 bit per pixel
SQ_TP2_lod_correct	SQ→TP2	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP2_pix_mask	SQ→TP2	4	Pixel mask 1 bit per pixel
SQ_TP3_lod_correct	SQ→TP3	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP3_pix_mask	SQ→TP3	4	Pixel mask 1 bit per pixel
SQ_TPx_rs_line_num	SQ→TPx	6	Line number in the Reservation station
SQ_TPx_write_gpr_index	SQ->TPx	7	Index into Register file for write of returned Fetch Data

~~24.2.9~~23.2.9 TP to SQ: Texture stall

Formatted: Bullets and Numbering

The TP sends this signal to the SQ and the SPs when its input buffer is full.



Name	Direction	Bits	Description
TP_SQ_fetch_stall	TP→ SQ	1	Do not send more texture request if asserted

~~24.2.10~~23.2.10 SQ to SP: Texture stall

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SPx_fetch_stall	SQ→SPx	1	Do not send more texture request if asserted

~~24.2.11~~23.2.11 SQ to SP: GPR and auto counter

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SPx_gpr_wr_addr	SQ→SPx	7	Write address
SQ_SPx_gpr_rd_addr	SQ→SPx	7	Read address
SQ_SPx_gpr_rd_en	SQ→SPx	1	Read Enable
SQ_SP0_gpr_wr_en	SQ→SPx	1	Write Enable for the GPRs of SP0
SQ_SP1_gpr_wr_en	SQ→SPx	1	Write Enable for the GPRs of SP1
SQ_SP2_gpr_wr_en	SQ→SPx	1	Write Enable for the GPRs of SP2
SQ_SP3_gpr_wr_en	SQ→SPx	1	Write Enable for the GPRs of SP3
SQ_SPx_gpr_phase	SQ→SPx	2	The phase mux (arbitrates between inputs, ALU SRC reads and writes)
SQ_SPx_channel_mask	SQ→SPx	4	The channel mask
SQ_SPx_gpr_input_sel	SQ→SPx	2	When the phase mux selects the inputs this tells from which source to read from: Interpolated data, VTX0, VTX1, autogen counter.
SQ_SPx_auto_count	SQ→SPx	12?	Auto count generated by the SQ, common for all shader pipes

~~24.2.12~~23.2.12 SQ to SPx: Instructions

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SPx_instr_start	SQ→SPx	1	Instruction start
SQ_SP_instr	SQ→SPx	22~~1~~	Transferred over 4 cycles 0: SRC A Select 2:0 SRC A Argument Modifier 3:3 SRC A swizzle 11:4 VectorDst 17:12 ~~Unused~~ Per channel use mask (PV/Reg) ~~20~~21:18 -- 1: SRC B Select 2:0 SRC B Argument Modifier 3:3 SRC B swizzle 11:4 ScalarDst 17:12 Per channel use mask (PV/Reg) 21:18~~Unused 20:18~~ -- 2: SRC C Select 2:0 SRC C Argument Modifier 3:3 SRC C swizzle 11:4 Per channel use mask (PV/Reg) 21:18~~Unused 20:12~~ -- 3: Vector Opcode 4:0 Scalar Opcode 10:5 Vector Clamp 11:11 Scalar Clamp 12:12 Vector Write Mask 16:13 Scalar Write Mask 20:17
SQ_SPx_exp_alu_id	SQ→SPx	1	ALU ID
SQ_SPx_exporting	SQ→SPx	2	0: Not Exporting 1: Vector Exporting 2: Scalar Exporting
SQ_SPx_stall	SQ→SPx	1	Stall signal
SQ_SP0_write_mask	SQ→SP0	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP1_ write_mask	SQ→SP1	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP2_ write_mask	SQ→SP2	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP3_ write_mask	SQ→SP3	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SPx_last	SQ→SPx	1	Last instruction of the block
SQ_SP0_pred_overwrite	SQ→SP0	4	Indicates to overwrite the use of PV/PS because of the predication (use the GPRs instead). This operation is done on a per-pixel basis.
SQ_SP1_pred_overwrite	SQ→SP1	4	Indicates to overwrite the use of PV/PS because of

			the predication (use the GPRs instead). This operation is done on a per-pixel basis.
SQ_SP2_pred_overwrite	SQ→SP2	4	Indicates to overwrite the use of PV/PS because of the predication (use the GPRs instead). This operation is done on a per-pixel basis.
SQ_SP3_pred_overwrite	SQ→SP3	4	Indicates to overwrite the use of PV/PS because of the predication (use the GPRs instead). This operation is done on a per-pixel basis.

~~24.2.13~~23.2.13 *SP to SQ: Constant address load/ Predicate Set*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SP0_SQ_const_addr	SP0→SQ	36	Constant address load / predicate vector load (4 bits only) to the sequencer
SP0_SQ_valid	SP0→SQ	1	Data valid
SP1_SQ_const_addr	SP1→SQ	36	Constant address load / predicate vector load (4 bits only) to the sequencer
SP1_SQ_valid	SP1→SQ	1	Data valid
SP2_SQ_const_addr	SP2→SQ	36	Constant address load / predicate vector load (4 bits only) to the sequencer
SP2_SQ_valid	SP2→SQ	1	Data valid
SP3_SQ_const_addr	SP3→SQ	36	Constant address load / predicate vector load (4 bits only) to the sequencer
SP3_SQ_valid	SP3→SQ	1	Data valid
SP0_SQ_data_type	SP→SQ	1	Data Type 0: Constant Load 1: Predicate Set

~~24.2.14~~23.2.14 *SQ to SPx: constant broadcast*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SPx_const	SQ→SPx	128	Constant broadcast

~~24.2.15~~23.2.15 *SP0 to SQ: Kill vector load*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SP0_SQ_kill_vect	SP0→SQ	4	Kill vector load
SP1_SQ_kill_vect	SP1→SQ	4	Kill vector load
SP2_SQ_kill_vect	SP2→SQ	4	Kill vector load
SP3_SQ_kill_vect	SP3→SQ	4	Kill vector load

~~24.2.16~~23.2.16 *SQ to CP: RBBM bus*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_RBB_rs	SQ→CP	1	Read Strobe
SQ_RBB_rd	SQ→CP	32	Read Data
SQ_RBBM_nrtrtr	SQ→CP	1	Optional
SQ_RBBM_rtr	SQ→CP	1	Real-Time (Optional)

~~24.2.17~~23.2.17 *CP to SQ: RBBM bus*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
rbbm_we	CP→SQ	1	Write Enable
rbbm_a	CP→SQ	15	Address -- Upper Extent is TBD (16:2)
rbbm_wd	CP→SQ	32	Data
rbbm_be	CP→SQ	4	Byte Enables
rbbm_re	CP→SQ	1	Read Enable
rbb_rs0	CP→SQ	1	Read Return Strobe 0
rbb_rs1	CP→SQ	1	Read Return Strobe 1
rbb_rd0	CP→SQ	32	Read Data 0

rbb_rd1	CP→SQ	32	Read Data 0
RBBM_SQ_soft_reset	CP→SQ	1	Soft Reset

~~24.2.18~~23.2.18 *SQ to CP: State report*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_CP_vs_event	SQ→CP	1	Vertex Shader Event
SQ_CP_vs_eventid	SQ→CP	2	Vertex Shader Event ID
SQ_CP_ps_event	SQ→CP	1	Pixel Shader Event
SQ_CP_ps_eventid	SQ→CP	2	Pixel Shader Event ID

eventid = 0 => *sEndOfState (i.e. VsEndOfState)
eventid = 1 => *sDone (i.e. VsDone)

So, the CP will assume the Vs is done with a state whenever it gets a pulse on the SQ_CP_vs_event and the SQ_CP_vs_eventid = 0.

~~24.3~~23.3 Example of control flow program execution

Formatted: Bullets and Numbering

We now provide some examples of execution to better illustrate the new design.

Given the program:

Alu 0
Alu 1
Tex 0
Tex 1
Alu 3 Serial
Alu 4
Tex 2
Alu 5
Alu 6 Serial
Tex 3
Alu 7
Alloc Position 1 buffer
Alu 8 Export
Tex 4
Alloc Parameter 3 buffers
Alu 9 Export 0
Tex 5
Alu 10 Serial Export 2
Alu 11 Export 1 End

Would be converted into the following CF instructions:

```
Execute ~~Alu ~~0 Alu ~~Alu ~~0 Alu ~~Tex ~~0 Tex ~~Tex ~~0 Tex ~~Alu ~~1 Alu ~~Alu ~~0 Alu ~~Tex ~~0 Tex ~~Alu ~~0
Alu ~~Alu ~~1 Alu ~~Tex ~~0 Tex
Execute ~~Alu ~~0 Alu
Alloc Position 1
Execute ~~Alu ~~0 Alu ~~Tex ~~0 Tex
Alloc Param 3
Execute_end ~~Alu ~~0 Alu ~~Tex ~~0 Tex ~~Alu ~~1 Alu ~~Alu ~~0 Alu ~~End~~
```

And the execution of this program would look like this:

Put thread in Vertex RS:

Control Flow Instruction Pointer (12 bits), (CFP)
Execution Count Marker (3 or 4 bits), (ECM)

Loop Iterators (4x9 bits), (LI)
Call return pointers (4x12 bits), (CRP)
Predicate Bits(4x64 bits), (PB)
Export ID (1 bit), (EXID)
GPR Base Ptr (8 bits), (GPR)
Export Base Ptr (7 bits), (EB)
Context Ptr (3 bits).(CPTR)
LOD correction bits (16x6 bits) (LOD)

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	0	0	0	0	0	0	0	0	0

Valid Thread (VALID)
Texture/ALU engine needed (TYPE)
Texture Reads are outstanding (PENDING)
Waiting on Texture Read to Complete (SERIAL)
Allocation Wait (2 bits) (ALLOC)
    00 – No allocation needed
    01 – Position export allocation needed (ordered export)
    10 – Parameter or pixel export needed (ordered export)
    11 – pass thru (out of order export)
Allocation Size (4 bits) (SIZE)
Position Allocated (POS_ALLOC)
First thread of a new context (FIRST)
Last (1 bit), (LAST)

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	0	0	0	0	0	1	0

Then the thread is picked up for the execution of the first control flow instruction:

Execute 0 Alu 0 Alu 0 Tex 0 Tex 1 Alu 0 Alu 0 Tex 0 Alu 1 Alu 0 Tex
~~Execute Alu 0 Alu 0 Tex 0 Tex 0 Alu 1 Alu 0 Tex 0 Alu 0 Alu 1 Tex 0~~

It executes the first two ALU instructions and goes back to the RS for a resource request change. Here is the state returned to the RS:

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	2	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	0	0	0	0	0	1	0

Then when the texture pipe frees up, the arbiter picks up the thread to issue the texture reads. The thread comes back in this state:

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	4	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST

1	ALU	1	1	0	0	0	1	0

Because of the serial bit the arbiter must wait for the texture to return and clear the PENDING bit before it can pick the thread up. Lets say that the texture reads are complete, then the arbiter picks up the thread and returns it in this state:

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	6	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	0	0	0	0	0	1	0

Again the TP frees up, the arbiter picks up the thread and executes. It returns in this state:

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	7	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	0	0	0	1	0

Now, even if the texture has not returned we can still pick up the thread for ALU execution because the serial bit is not set. The thread will however come back to the RS for the second ALU instruction because it has the serial bit set.

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	8	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	1	0	0	0	1	0

As soon as the TP clears the pending bit the thread is picked up and returns:

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	9	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	0	0	0	0	0	1	0

Picked up by the TP and returns:

Execute 0 Alu

~~Execute Alu 0~~

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
1	0	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	0	0	0	1	0

Picked up by the ALU and returns (lets say the TP has not returned yet):

```
Alloc Position 1
```

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
2	0	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	01	1	0	1	0

If the SX has the place for the export, the SQ is going to allocate and pick up the thread for execution. It returns to the RS in this state:

Execute ~~Alu ~~0 Alu ~~Tex ~~0 Tex

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
3	1	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	1	0	0	0	1	1	0

Now, since the TP has not returned yet, we must wait for it to return because we cannot issue multiple texture requests. The TP returns, clears the PENDING bit and we proceed:

```
Alloc Param 3
```

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
4	0	0	0	0	1	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	10	3	1	1	0

Once again the SQ makes sure the SX has enough room in the Parameter cache before it can pick up this thread.

Execute_end ~~Alu ~~0 Alu ~~Tex ~~0 Tex ~~Alu ~~1 Alu ~~Alu ~~0 Alu~~End~~

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
5	1	0	0	0	1	0	100	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	1	0	0	0	1	1	0

This executes on the TP and then returns:

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
5	2	0	0	0	1	0	100	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	1	0	0	1	1	1

Waits for the TP to return because of the textures reads are pending (and SERIAL in this case). Then executes and does not return to the RS because the LAST bit is set. This is the end of this thread and before dropping it on the floor, the SQ notifies the SX of export completion.

Formatted: Bullets and Numbering

~~25.~~24. Open issues

Need to do some testing on the size of the register file as well as on the register file allocation method (dynamic VS static).

Saving power?

	ORIGINATE DATE 24 September, 2001	EDIT DATE 4 September, 2015~~2~~ August, 200215 July	DOCUMENT-REV. NUM. GEN-CXXXXX-REVA	PAGE 1 of 51

Author: Laurent Lefebvre

Issue To:	**Copy No:**

R400 Sequencer Specification

SQ

Version 2.04

Overview: This is an architectural specification for the R400 Sequencer block (SEQ). It provides an overview of the required capabilities and expected uses of the block. It also describes the block interfaces, internal sub-blocks, and provides internal state diagrams.

AUTOMATICALLY UPDATED FIELDS:

Document Location:	C:\perforce\r400\doc_lib\design\blocks\sq\R400_Sequencer.doc
Current Intranet Search Title:	R400 Sequencer Specification

APPROVALS	
Name/Dept	Signature/Date

Remarks:

THIS DOCUMENT CONTAINS CONFIDENTIAL INFORMATION THAT COULD BE SUBSTANTIALLY DETRIMENTAL TO THE INTEREST OF ATI TECHNOLOGIES INC. THROUGH UNAUTHORIZED USE OR DISCLOSURE.

ATI 2032
LG v. ATI
IPR2015-00326

Table Of Contents

1. OVERVIEW 7
1.1 Top Level Block Diagram 9
1.2 Data Flow graph (SP) 10
1.3 Control Graph 11
2. INTERPOLATED DATA BUS 11
3. INSTRUCTION STORE 14
4. SEQUENCER INSTRUCTIONS 14
5. CONSTANT STORES 14
5.1 Memory organizations 14
5.2 Management of the Control Flow Constants 15
5.3 Management of the re-mapping tables 15
5.3.1 R400 Constant management 15
5.3.2 Proposal for R400LE constant management 15
5.3.3 Dirty bits 17
5.3.4 Free List Block 17
5.3.5 De-allocate Block 18
5.3.6 Operation of Incremental model 18
5.4 Constant Store Indexing 18
5.5 Real Time Commands 19
5.6 Constant Waterfalling 19
6. LOOPING AND BRANCHES 20
6.1 The controlling state. 20
6.2 The Control Flow Program 20
6.2.1 Control flow instructions table 21
6.3 Implementation 23
6.4 Data dependant predicate instructions 24
6.5 HW Detection of PV,PS 25
6.6 Register file indexing 25
6.7 Debugging the Shaders .... 26~~25~~
6.7.1 Method 1: Debugging registers 26
6.7.2 Method 2: Exporting the values in the GPRs 26
7. PIXEL KILL MASK 26
8. MULTIPASS VERTEX SHADERS (HOS) 26
9. REGISTER FILE ALLOCATION .... 27~~26~~
10. FETCH ARBITRATION 28
11. ALU ARBITRATION 28
12. HANDLING STALLS 29
13. CONTENT OF THE RESERVATION STATION FIFOS 29
14. THE OUTPUT FILE 29
15. IJ FORMAT 29
15.1 Interpolation of constant attributes 29
16. STAGING REGISTERS 30

17. THE PARAMETER CACHE 31
17.1 Export restrictions 32
17.1.1 Pixel exports: 32
17.1.2 Vertex exports: 32
17.1.3 Pass thru exports: 32
17.2 Arbitration restrictions 32
18. EXPORT TYPES 32
18.1 Vertex Shading 32
18.2 Pixel Shading 33
19. SPECIAL INTERPOLATION MODES 33
19.1 Real time commands 33
19.2 Sprites/ XY screen coordinates/ FB information 33
19.3 Auto generated counters 34
19.3.1 Vertex shaders 34
19.3.2 Pixel shaders 34
20. STATE MANAGEMENT .... 35~~34~~
20.1 Parameter cache synchronization .... 35~~34~~
21. XY ADDRESS IMPORTS 35
21.1 Vertex indexes imports 35
22. REGISTERS 35
22.1 Control .... **Error! Bookmark not defined.~~Error! Bookmark not defined.~~**~~35~~
22.2 Context .... **Error! Bookmark not defined.~~Error! Bookmark not defined.~~**~~35~~
23. DEBUG REGISTERS ..... ERROR! BOOKMARK NOT DEFINED.~~ERROR! BOOKMARK NOT DEFINED.~~**~~35~~**
23.1 Context .... **Error! Bookmark not defined.~~Error! Bookmark not defined.~~**~~35~~
23.2 Control .... **Error! Bookmark not defined.~~Error! Bookmark not defined.~~**~~35~~
24. INTERFACES .... 36~~35~~
24.1 External Interfaces .... 36~~35~~
24.2 SC to SP Interfaces .... 36~~35~~
24.2.1 SC_SP# .... 36~~35~~
24.2.2 SC_SQ .... 37~~36~~
24.2.3 SQ to SX: Interpolator bus 38
24.2.4 SQ to SP: Staging Register Data 38
24.2.5 VGT to SQ : Vertex interface .... 39~~38~~
24.2.6 SQ to SX: Control bus .... 42~~41~~
24.2.7 SX to SQ : Output file control .... 42~~41~~
24.2.8 SQ to TP: Control bus .... 43~~42~~
24.2.9 TP to SQ: Texture stall .... 43~~42~~
24.2.10 SQ to SP: Texture stall .... 44~~43~~
24.2.11 SQ to SP: GPR and auto counter .... 44~~43~~
24.2.12 SQ to SPx: Instructions .... 45~~44~~
24.2.13 SP to SQ: Constant address load/ Predicate Set .... 46~~45~~44

24.2.14 SQ to SPx: constant broadcast .... 46~~45~~
24.2.15 SP0 to SQ: Kill vector load .... 46~~45~~
24.2.16 SQ to CP: RBBM bus .... 46~~45~~
24.2.17 CP to SQ: RBBM bus .... 46~~45~~
24.2.18 SQ to CP: State report .... 47~~45~~
24.3 Example of control flow program execution .... 47~~46~~
25. OPEN ISSUES .... 51~~50~~

Revision Changes:

Rev 0.1 (Laurent Lefebvre) Date: May 7, 2001	First draft.
Rev 0.2 (Laurent Lefebvre) Date : July 9, 2001	Changed the interfaces to reflect the changes in the SP. Added some details in the arbitration section.
Rev 0.3 (Laurent Lefebvre) Date : August 6, 2001	Reviewed the Sequencer spec after the meeting on August 3, 2001.
Rev 0.4 (Laurent Lefebvre) Date : August 24, 2001	Added the dynamic allocation method for register file and an example (written in part by Vic) of the flow of pixels/vertices in the sequencer.
Rev 0.5 (Laurent Lefebvre) Date : September 7, 2001	Added timing diagrams (Vic)
Rev 0.6 (Laurent Lefebvre) Date : September 24, 2001	Changed the spec to reflect the new R400 architecture. Added interfaces.
Rev 0.7 (Laurent Lefebvre) Date : October 5, 2001	Added constant store management, instruction store management, control flow management and data dependant predication.
Rev 0.8 (Laurent Lefebvre) Date : October 8, 2001	Changed the control flow method to be more flexible. Also updated the external interfaces.
Rev 0.9 (Laurent Lefebvre) Date : October 17, 2001	Incorporated changes made in the 10/18/01 control flow meeting. Added a NOP instruction, removed the conditional_execute_or_jump. Added debug registers.
Rev 1.0 (Laurent Lefebvre) Date : October 19, 2001	Refined interfaces to RB. Added state registers.
Rev 1.1 (Laurent Lefebvre) Date : October 26, 2001	Added SEQ→SP0 interfaces. Changed delta precision. Changed VGT→SP0 interface. Debug Methods added.
Rev 1.2 (Laurent Lefebvre) Date : November 16, 2001	Interfaces greatly refined. Cleaned up the spec.
Rev 1.3 (Laurent Lefebvre) Date : November 26, 2001	Added the different interpolation modes.
Rev 1.4 (Laurent Lefebvre) Date : December 6, 2001	Added the auto incrementing counters. Changed the VGT→SQ interface. Added content on constant management. Updated GPRs.
Rev 1.5 (Laurent Lefebvre) Date : December 11, 2001	Removed from the spec all interfaces that weren't directly tied to the SQ. Added explanations on constant management. Added PA→SQ synchronization fields and explanation.
Rev 1.6 (Laurent Lefebvre) Date : January 7, 2002	Added more details on the staging register. Added detail about the parameter caches. Changed the call instruction to a Conditionnal_call instruction. Added details on constant management and updated the diagram.
Rev 1.7 (Laurent Lefebvre) Date : February 4, 2002	Added Real Time parameter control in the SX interface. Updated the control flow section.
Rev 1.8 (Laurent Lefebvre) Date : March 4, 2002	New interfaces to the SX block. Added the end of clause modifier, removed the end of clause instructions.
Rev 1.9 (Laurent Lefebvre) Date : March 18, 2002	Rearangement of the CF instruction bits in order to ensure byte alignement.
Rev 1.10 (Laurent Lefebvre) Date : March 25, 2002	Updated the interfaces and added a section on exporting rules.
Rev 1.11 (Laurent Lefebvre) Date : April 19, 2002	Added CP state report interface. Last version of the spec with the old control flow scheme
Rev 2.0 (Laurent Lefebvre) Date : April 19, 2002	New control flow scheme

Rev 2.01 (Laurent Lefebvre) Date : May 2, 2002	Changed slightly the control flow instructions to allow force jumps and calls.
Rev 2.02 (Laurent Lefebvre) Date : May 13, 2002	Updated the Opcodes. Added type field to the constant/pred interface. Added Last field to the SQ→SP instruction load interface.
Rev 2.03 (Laurent Lefebvre) Date : July 15, 2002	SP interface updated to include predication optimizations. Added the predicate no stall instructions,
Rev 2.04 (Laurent Lefebvre) Date :August 2, 2002	Documented the new parameter generation scheme for XY coordinates points and lines STs.

1. Overview

The sequencer chooses two ALU threads and a fetch hread to execute, and executes all of the instructions in a block before looking for a new clause of the same type. Two ALU threads are executed interleaved to hide the ALU latency. The arbitrator will give priority to older threads. There are two separate reservation stations, one for pixel vectors and one for vertices vectors. This way a pixel can pass a vertex and a vertex can pass a pixel.

To support the shader pipe the sequencer also contains the shader instruction cache, constant store, control flow constants and texture state. The four shader pipes also execute the same instruction thus there is only one sequencer for the whole chip.

The sequencer first arbitrates between vectors of 64 vertices that arrive directly from primitive assembly and vectors of 16 quads (64 pixels) that are generated in the scan converter.

The vertex or pixel program specifies how many GPRs it needs to execute. The sequencer will not start the next vector until the needed space is available in the GPRs.

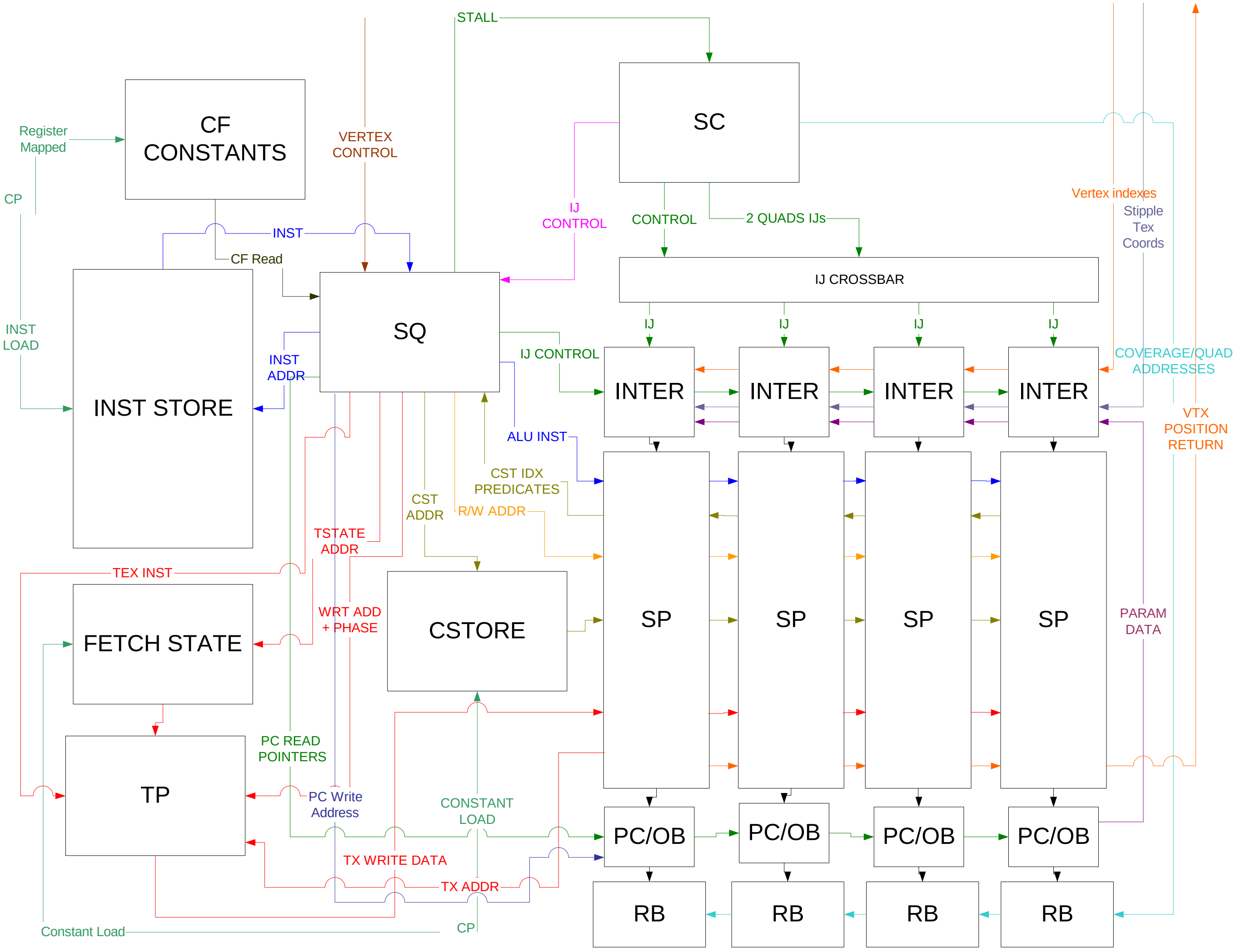


Figure 1: General Sequencer overview

1.1 Top Level Block Diagram

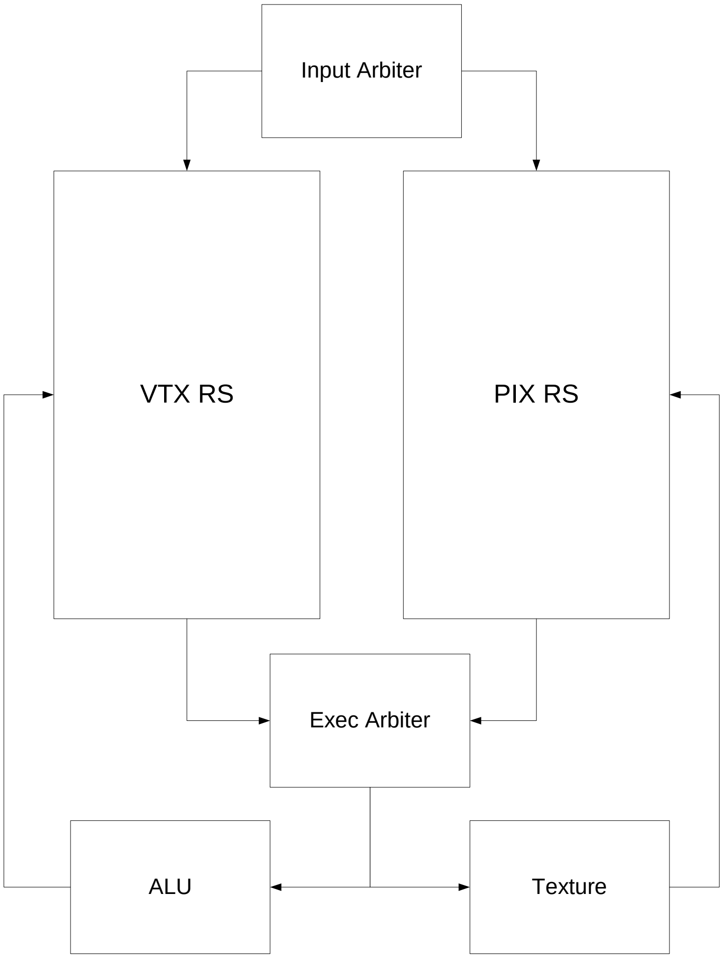


Figure 2: Reservation stations and arbiters

Under this new scheme, the sequencer (SQ) will only use one global state management machine per vector type (pixel, vertex) that we call the reservation station (RS).

1.2 Data Flow graph (SP)

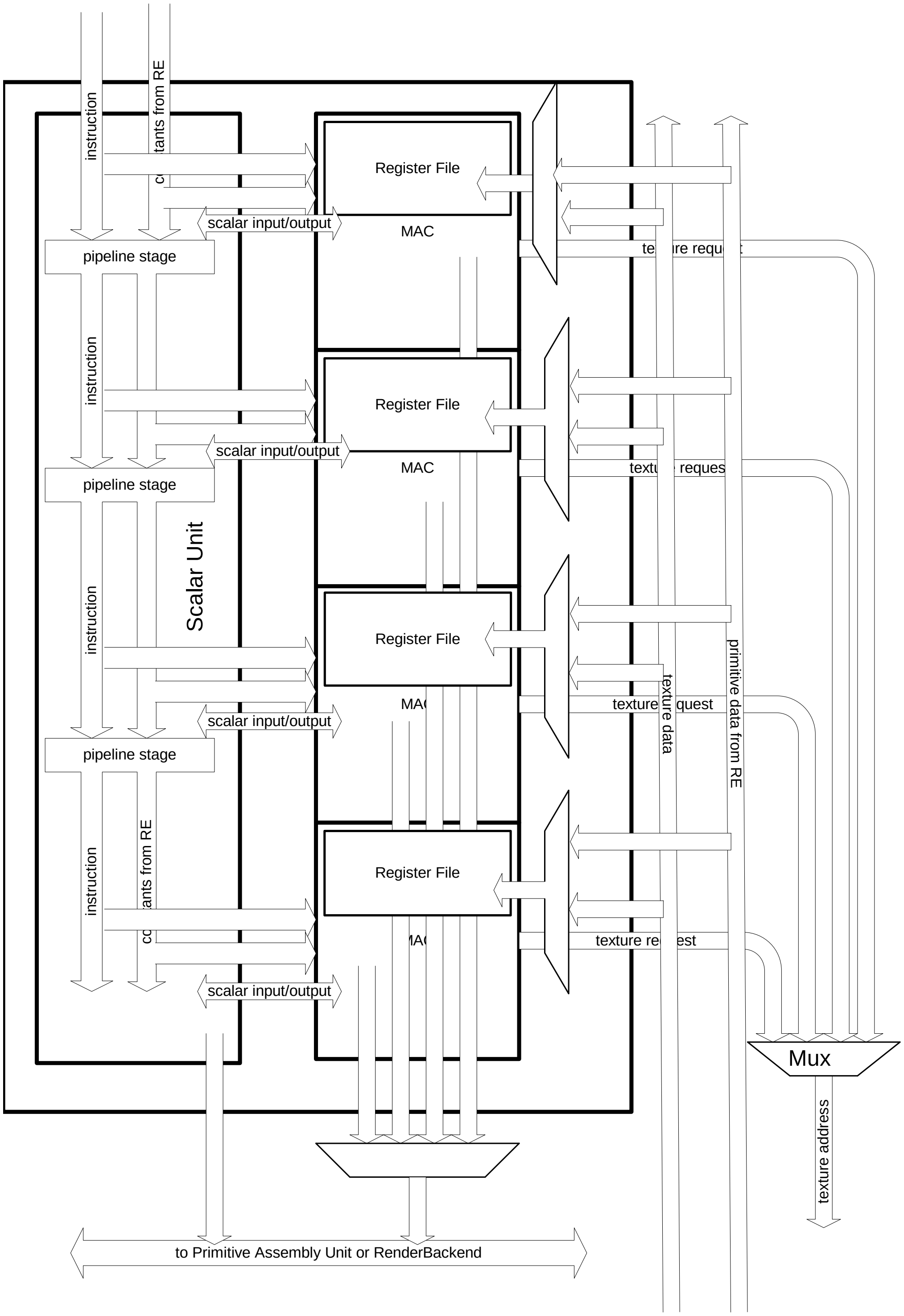


Figure 3: The shader Pipe

The gray area represents blocks that are replicated 4 times per shader pipe (16 times on the overall chip).

1.3 Control Graph

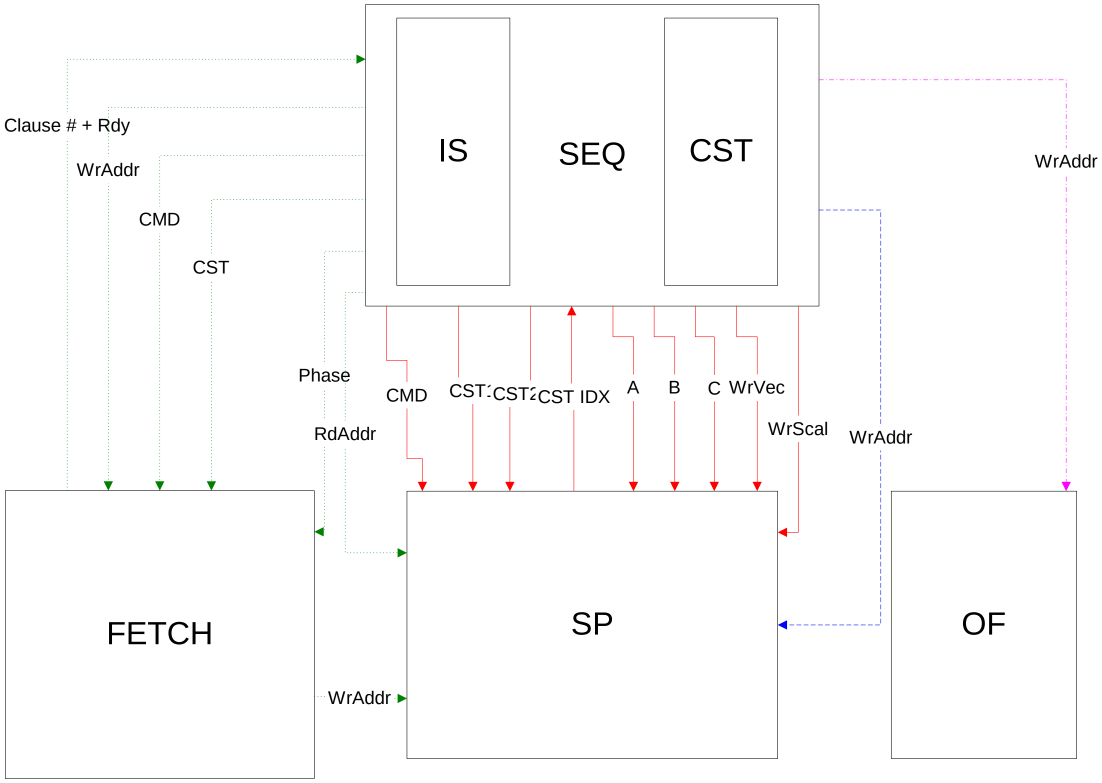


Figure 4: Sequencer Control interfaces

In green is represented the Fetch control interface, in red the ALU control interface, in blue the Interpolated/Vector control interface and in purple is the output file control interface.

2. Interpolated data bus

The interpolators contain an IJ buffer to pack the information as much as possible before writing it to the register file.

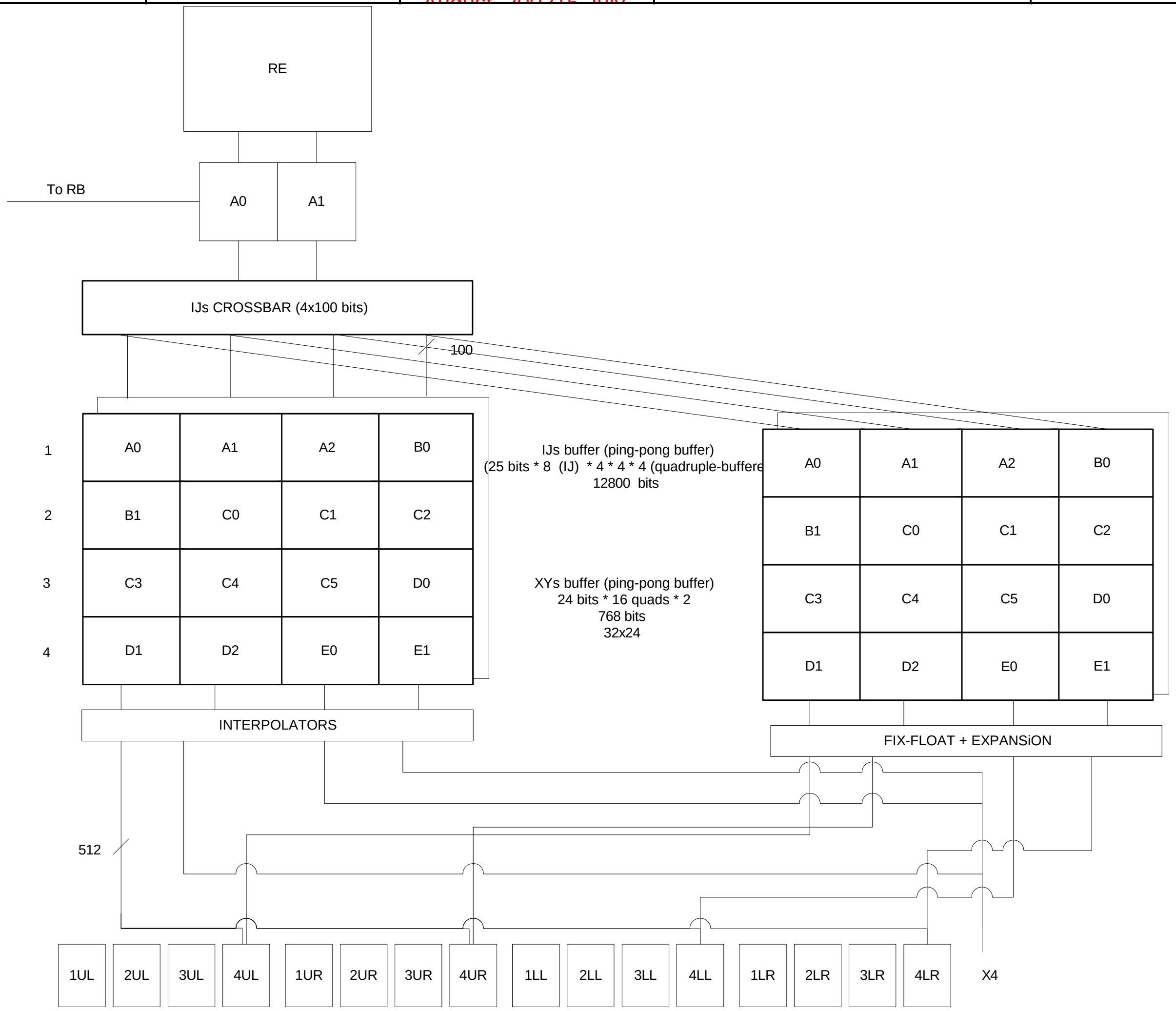


Figure 5: Interpolation buffers

WRITES

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21	T22	T23
SP 0	A0	A0	XY A0	B1	B1	XY B1	C3	C3	XY C3				D1	D1	XY D1									
SP 1	A1	A1	XY A1				C0	C0	XY C0	C4	C4	XY C4	D2	D2	XY D2									
SP 2	A2	A2	XY A2				C1	C1	XY C1	C5	C5	XY C5				E0	E0	XY E0						
SP 3				B0	B0	XY B0	C2	C2	XY C2				D0	D0	XY D0	E1	E1	XY E1						

READS

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21	T22	T23
SP 0	XY 0-3	XY 16-19	XY 32-35	XY 48-51	A0	B1	C3	D1					A0	B1	C3	D1					V 0-3	V 16-19	V 32-35	V 48-51
SP 1	XY 4-7	XY 20-23	XY 36-39	XY 52-55	A1		C4	D2		C0			A1		C4	D2		C0			V 4-7	V 20-23	V 36-39	V 52-55
SP 2	XY 8-11	XY 24-27	XY 40-43	XY 56-59	A2		C5			C1		E0	A2		C5			C1		E0	V 8-11	V 24-27	V 40-43	V 56-59
SP 3	XY 12-15	XY 28-31	XY 44-47	XY 60-63					B0	C2	D0	E1					B0	C2	D0	E1	V 12-15	V 28-31	V 44-47	V 60-63
	XY				P1								P2								VTX			

Figure 6: Interpolation timing diagram

Above is an example of a tile the sequencer might receive from the SC. The write side is how the data get stacked into the XY and IJ buffers, the read side is how the data is passed to the GPRs. The IJ information is packed in the IJ buffer 4 quads at a time or two clocks. The sequencer allows at any given time as many as four quads to interpolate a parameter. They all have to come from the same primitive. Then the sequencer controls the write mask to the GPRs to write the valid data in.

3. Instruction Store

There is going to be only one instruction store for the whole chip. It will contain 4096 instructions of 96 bits each.

It is likely to be a 1 port memory; we use 1 clock to load the ALU instruction, 1 clocks to load the Fetch instruction, 1 clock to load 2 control flow instructions and 1 clock to write instructions.

The instruction store is loaded by the CP thru the register mapped registers.

The VS_BASE and PS_BASE context registers are used to specify for each context where its shader is in the instruction memory.

For the Real time commands the story is quite the same but for some small differences. There are no wrap-around points for real time so the driver must be careful not to overwrite regular shader data. The shared code (shared subroutines) uses the same path as real time.

4. Sequencer Instructions

All control flow instructions and move instructions are handled by the sequencer only. The ALUs will perform NOPs during this time (MOV PV,PV, PS,PS) if they have nothing else to do.

5. Constant Stores

5.1 Memory organizations

A likely size for the ALU constant store is 1024x128 bits. The read BW from the ALU constant store is 128 bits/clock and the write bandwidth is 32 bits/clock (directed by the CP bus size not by memory ports).

The maximum logical size of the constant store for a given shader is 256 constants. Or 512 for the pixel/vertex shader pair. The size of the re-mapping table is 128 lines (each line addresses 4 constants). The write granularity is 4 constants or 512 bits. It takes 16 clocks to write the four constants. Real time requires 256 lines in the physical memory (this is physically register mapped).

The texture state is also kept in a similar memory. The size of this memory is 320x96 bits (128 texture states for regular mode, 32 states for RT). The memory thus holds 128 texture states (192 bits per state). The logical size exposes 32 different states total, which are going to be shared between the pixel and the vertex shader. The size of the re-mapping table to for the texture state memory is 32 lines (each line addresses 1 texture state lines in the real memory). The CP write granularity is 1 texture state lines (or 192 bits). The driver sends 512 bits but the CP ignores the top 320 bits. It thus takes 6 clocks to write the texture state. Real time requires 32 lines in the physical memory (this is physically register mapped).

The control flow constant memory doesn't sit behind a renaming table. It is register mapped and thus the driver must reload its content each time there is a change in the control flow constants. Its size is 320*32 because it must hold 8 copies of the 32 dwords of control flow constants and the loop construct constants must be aligned.

The constant re-mapping tables for texture state and ALU constants are logically register mapped for regular mode and physically register mapped for RT operation.

5.2 Management of the Control Flow Constants

The control flow constants are register mapped, thus the CP writes to the according register to set the constant, the SQ decodes the address and writes to the block pointed by its current base pointer (CF_WR_BASE). On the read side, one level of indirection is used. A register (SQ_CONTEXT_MISC.CF_RD_BASE) keeps the current base pointer to the control flow block. This register is copied whenever there is a state change. Should the CP write to CF after the state change, the base register is updated with the (current pointer number +1)% number of states. This way, if the CP doesn't write to CF the state is going to use the previous CF constants.

5.3 Management of the re-mapping tables

5.3.1 *R400 Constant management*

The sequencer is responsible to manage two re-mapping tables (one for the constant store and one for the texture state). On a state change (by the driver), the sequencer will broadside copy the contents of its re-mapping tables to a new one. We have 8 different re-mapping tables we can use concurrently.

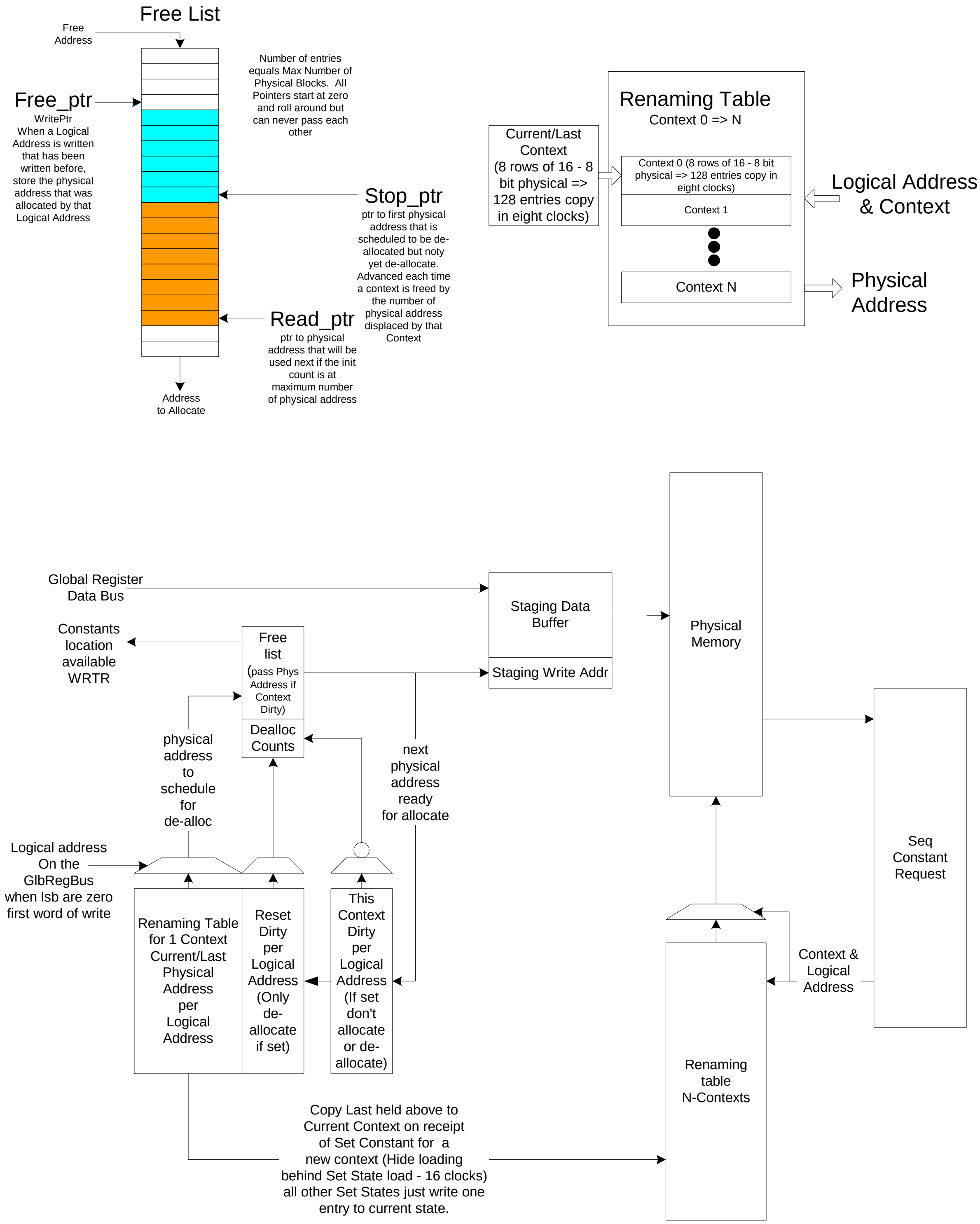
The constant memory update will be incremental, the driver only need to update the constants that actually changed between the two state changes.

For this model to work in its simplest form, the requirement is that the physical memory MUST be at least twice as large as the logical address space + the space allocated for Real Time. In our case, since the logical address space is 512 and the reserved RT space can be up to 256 entries, the memory must be of sizes 1280 and above. Similarly the size of the texture store must be of 32*2+32 = 96 entries and above.

5.3.2 *Proposal for R400LE constant management*

To make this scheme work with only 512+256 = 768 entries, upon reception of a CONTROL packet of state + 1, the sequencer would check for SQ_IDLE and PA_IDLE and if both are idle will erase the content of state to replace it with the new state (this is depicted in Figure 8: De-allocation mechanism~~Figure 8: De-allocation mechanism~~~~Figure 8: De-allocation mechanism~~). Note that in the case a state is cleared a value of 0 is written to the corresponding de-allocation counter location so that when the SQ is going to report a state change, nothing will be de-allocated upon the first report.

The second path sets all context dirty bits that were used in the current state to 1 (thus allowing the new state to reuse these physical addresses if needed).



Free List
Free Address
Free_ptr
WritePtr When a Logical Address is written that has been written before, store the physical address that was allocated by that Logical Address
Number of entries equals Max Number of Physical Blocks. All Pointers start at zero and roll around but can never pass each other
Stop_ptr
ptr to first physical address that is scheduled to be de-allocated but noty yet de-allocate. Advanced each time a context is freed by the number of physical address displaced by that Context
Read_ptr
ptr to physical address that will be used next if the init count is at maximum number of physical address
Address to Allocate
Renaming Table
Context 0 => N
Current/Last Context (8 rows of 16 - 8 bit physical => 128 entries copy in eight clocks)
Context 0 (8 rows of 16 - 8 bit physical => 128 entries copy in eight clocks)
Context 1
Context N
Logical Address & Context
Physical Address
Global Register Data Bus
Staging Data Buffer
Staging Write Addr
Physical Memory
Constants location available WRTR
Free list (pass Phys Address if Context Dirty)
Dealloc Counts
physical address to schedule for de-alloc
next physical address ready for allocate
Seq Constant Request
Logical address On the GlbRegBus when lsb are zero first word of write
Renaming Table for 1 Context Current/Last Physical Address per Logical Address
Reset Dirty per Logical Address (Only de-allocate if set)
This Context Dirty per Logical Address (If set don't allocate or de-allocate)
Context & Logical Address
Renaming table N-Contexts
Copy Last held above to Current Context on receipt of Set Constant for a new context (Hide loading behind Set State load - 16 clocks) all other Set States just write one entry to current state.

Figure 7: Constant management

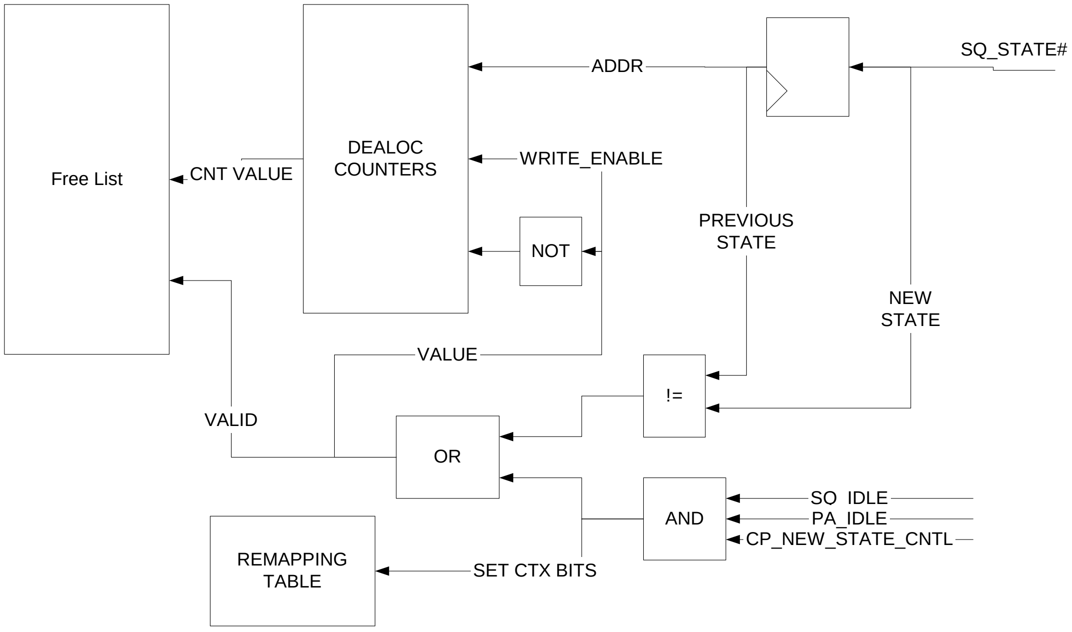


Figure 8: De-allocation mechanism for R400LE

5.3.3 *Dirty bits*

Two sets of dirty bits will be maintained per logical address. The first one will be set to zero on reset and set when the logical address is addressed. The second one will be set to zero whenever a new context is written and set for each address written while in this context. The reset dirty is not set, then writing to that logical address will not require de-allocation of whatever address stored in the renaming table. If it is set and the context dirty is not set, then the physical address store needs to be de-allocated and a new physical address is necessary to store the incoming data. If they are both set, then the data will be written into the physical address held in the renaming for the current logical address. No de-allocation or allocation takes place. This will happen when the driver does a set constant twice to the same logical address between context changes. NOTE: It is important to detect and prevent this, failure to do it will allow multiple writes to allocate all physical memory and thus hang because a context will not fit for rendering to start and thus free up space.

5.3.4 *Free List Block*

A free list block that would consist of a counter (called the IFC or Initial Free Counter) that would reset to zero and incremented every time a chunk of physical memory is used until they have all been used once. This counter would be checked each time a physical block is needed, and if the original ones have not been used up, us a new one, else check the free list for an available physical block address. The count is the physical address for when getting a chunk from the counter.

Storage of a free list big enough to store all physical block addresses.

Maintain three pointers for the free list that are reset to zero. The first one we will call write_ptr. This pointer will identify the next location to write the physical address of a block to be de-allocated. Note: we can never free more physical memory locations than we have. Once recording address the pointer will be incremented to walk the free list like a ring.

The second pointer will be called stop_ptr. The stop_ptr pointer will be advanced by the number of address chunks de-allocates when a context finishes. The address between the stop_ptr and write_ptr cannot be reused because they are still in use. But as soon as the context using then is dismissed the stop_ptr will be advanced.

The third pointer will be called read_ptr. This pointer will point will point to the next address that can be used for allocation as long as the read_ptr does not equal the stop_ptr and the IFC is at its maximum count.

5.3.5 *De-allocate Block*

This block will maintain a free physical address block count for each context. While in current context, a count shall be maintained specifying how many blocks were written into the free list at the write_ptr pointer. This count will be reset upon reset or when this context is active on the back and different than the previous context. It is actually a count of blocks in the previous context that will no longer be used. This count will be used to advance the write_ptr pointer to make available the set of physical blocks freed when the previous context was done. This allows the discard or de-allocation of any number of blocks in one clock.

5.3.6 *Operation of Incremental model*

The basic operation of the model would start with the write_ptr, stop_ptr, read_ptr pointers in the free list set to zero and the free list counter is set to zero. Also all the dirty bits and the previous context will be initialized to zero. When the first set constants happen, the reset dirty bit will not be set, so we will allocate a physical location from the free list counter because its not at the max value. The data will be written into physical address zero. Both the additional copy of the renaming table and the context zeros of the big renaming table will be updated for the logical address that was written by set start with physical address of 0. This process will be repeated for any logical address that are not dirty until the context changes. If a logical address is hit that has its dirty bits set while in the same context, both dirty bits would be set, so the new data will be over-written to the last physical address assigned for this logical address. When the first draw command of the context is detected, the previous context stored in the additional renaming table will be copied to the larger renaming table in the current (new) context location. Then the set constant logical address with be loaded with a new physical address during the copy and if the reset dirty was set, the physical address it replaced in the renaming table would be entered at the write_ptr pointer location on the free list and the write_ptr will be incremented. The de-allocation counter for the previous context (eight) will be incremented. This as set states come in for this context one of the following will happen:

1.) No dirty bits are set for the logical address being updated. A line will be allocated of the free-list counter or the free list at read_ptr pointer if read_ptr != to stop_ptr .
2.) Reset dirty set and Context dirty not set. A new physical address is allocated, the physical address in the renaming table is put on the free list at write_ptr and it is incremented along with the de-allocate counter for the last context.
3.) Context dirty is set then the data will be written into the physical address specified by the logical address.

This process will continue as long as set states arrive. This block will provide backpressure to the CP whenever he has not free list entries available (counter at max and stop_ptr == read_ptr). The command stream will keep a count of contexts of constants in use and prevent more than max constants contexts from being sent.

Whenever a draw packet arrives, the content of the re-mapping table is written to the correct re-mapping table for the context number. Also if the next context uses less constants than the current one all exceeding lines are moved to the free list to be de-allocated later. This happens in parallel with the writing of the re-mapping table to the correct memory.

Now preferable when the constant context leaves the last ALU clause it will be sent to this block and compared with the previous context that left. (Init to zero) If they differ than the older context will no longer be referenced and thus can be de-allocated in the physical memory. This is accomplished by adding the number of blocks freed this context to the stop_ptr pointer. This will make all the physical addresses used by this context available to the read_ptr allocate pointer for future allocation.

This device allows representation of multiple contexts of constants data with N copies of the logical address space. It also allows the second context to be represented as the first set plus some new additional data by just storing the delta's. It allows memory to be efficiently used and when the constants updates are small it can store multiple context. However, if the updates are large, less contexts will be stored and potentially performance will be degraded. Although it will still perform as well as a ring could in this case.

5.4 Constant Store Indexing

In order to do constant store indexing, the sequencer must be loaded first with the indexes (that come from the GPRs). There are 144 wires from the exit of the SP to the sequencer (9 bits pointers x 16 vertexes/clock). Since the data must pass thru the Shader pipe for the float to fixed conversion, there is a latency of 4 clocks (1 instruction)

between the time the sequencer is loaded and the time one can index into the constant store. The assembly will look like this

```
MOVA  R1.X,R2.X       // Loads the sequencer with the content of R2.X, also copies the content of R2.X into R1.X
NOP                   // latency of the float to fixed conversion
ADD   R3,R4,C0[R2.X]// Uses the state from the sequencer to add R4 to C0[R2.X] into R3
```

Note that we don’t really care about what is in the brackets because we use the state from the MOVA instruction. R2.X is just written again for the sake of simplicity and coherency.

The storage needed in the sequencer in order to support this feature is 2*64*9 bits = 1152 bits.

5.5 Real Time Commands

The real time commands constants are written by the CP using the register mapped registers allocated for RT. It works is the same way than when dealing with regular constant loads BUT in this case the CP is not sending a logical address but rather a physical address and the reads are not passing thru the re-mapping table but are directly read from the memory. The boundary between the two zones is defined by the CONST_EO_RT control register. Similarly, for the fetch state, the boundary between the two zones is defined by the TSTATE_EO_RT control register.

5.6 Constant Waterfalling

In order to have a reasonable performance in the case of constant store indexing using the address register, we are going to have the possibility of using the physical memory port for read only. This way we can read 1 constant per clock and thus have a worst-case waterfall mode of 1 vertex per clock. There is a small synchronization issue related with this as we need for the SQ to make sure that the constants where actually written to memory (not only sent to the sequencer) before it can allow the first vector of pixels or vertices of the state to go thru the ALUs. To do so, the sequencer keeps 8 bits (one per render state) and sets the bits whenever the last render state is written to memory and clears the bit whenever a state is freed.

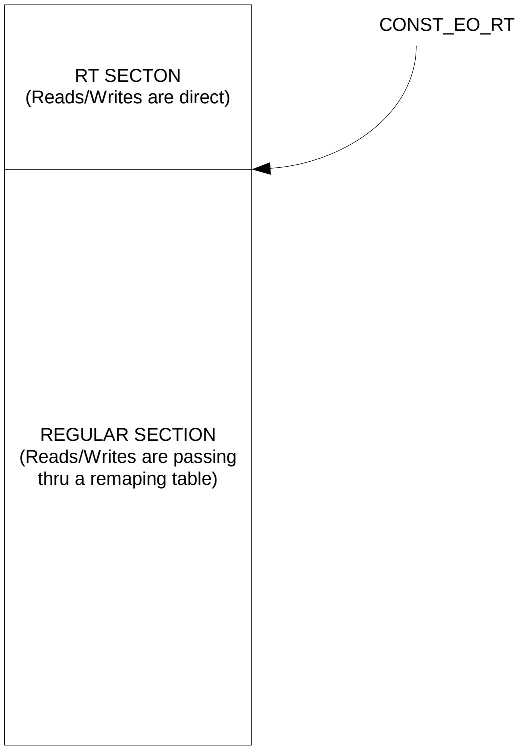


Figure 9: The Constant store

6. Looping and Branches

Loops and branches are planned to be supported and will have to be dealt with at the sequencer level. We plan on supporting constant loops and branches using a control program.

6.1 The controlling state.

The R400 controling state consists of:

Boolean[256:0]
Loop_count[7:0][31:0]
Loop_Start[7:0][31:0]
Loop_Step[7:0][31:0]

That is 256 Booleans and 32 loops.

We have a stack of 4 elements for nested calls of subroutines and 4 loop counters to allow for nested loops.

This state is available on a per shader program basis.

6.2 The Control Flow Program

We'd like to be able to code up a program of the form:

```
1:      Loop
2:      Exec    TexFetch
3:              TexFetch
4:              ALU
5:              ALU
6:              TexFetch
7:      End Loop
8:      ALU Export
```

But realize that 3: may be dependent on 2: and 4: is almost certainly dependent on 2: and 3:. Without clausing, these dependencies need to be expressed in the Control Flow instructions. Additionally, without separate 'texture clauses' and 'ALU clauses' we need to know which instructions to dispatch to the Texture Unit and which to the ALU unit. This information will be encapsulated in the flow control instructions.

Each control flow instruction will contain 2 bits of information for each (non-control flow) instruction:
a) ALU or Texture
b) Serialize Execution

(b) would force the thread to stop execution at this point (before the instruction is executed) and wait until all textures have been fetched. Given the allocation of reserved bits, this would mean that the count of an 'Exec' instruction would be limited to about 8 (non-control-flow) instructions. If more than this were needed, a second Exec (with the same conditions) would be issued.

Another function that relies upon 'clauses' is allocation and order of execution. We need to assure that pixels and vertices are exported in the correct order (even if not all execution is ordered) and that space in the output buffers are allocated in order. Additionally data can't be exported until space is allocated. A new control flow instruction:

Alloc <buffer select -- position,parameter, pixel or vertex memory. And the size required>.

would be created to mark where such allocation needs to be done. To assure allocation is done in order, the actual allocation for a given thread can not be performed unless the equivalent allocation for all previous threads is already completed. The implementation would also assure that execution of instruction(s) following the serialization due to the Alloc will occur in order -- at least until the next serialization or change from ALU to Texture. In most cases this will allow the exports to occur without any further synchronization. Only 'final' allocations or position allocations are

guaranteed to be ordered. Because strict ordering is required for pixels, parameters and positions, this implies only a single alloc for these structures. Vertex exports to memory do not require ordering during allocation and so multiple 'allocs' may be done.

6.2.1 *Control flow instructions table*

Here is the revised control flow instruction set.

Note that whenever a field is marked as RESERVED, it is assumed that all the bits of the field are cleared (0).

NOP		
47 … 44	43	42 … 0
0000	Addressing	RESERVED

This is a regular NOP.

Execute					
47 … 44	43	40 … 34	33 …16	15…12	11 … 0
0001	Addressing	RESERVED	Instructions type + serialize (9 instructions)	Count	Exec Address

Execute_End					
47 … 44	43	40 … 34	33 …16	15…12	11 … 0
0010	Addressing	RESERVED	Instructions type + serialize (9 instructions)	Count	Exec Address

Execute up to 9 instructions at the specified address in the instruction memory. The Instruction type field tells the sequencer the type of the instruction (LSB) (1 = Texture, 0 = ALU and whether to serialize or not the execution (MSB) (1 = Serialize, 0 = Non-Serialized). If Execute_End this is the last execution block of the shader program.

Conditional_Execute						
47 … 44	43	42	41 … 34	33…16	15 …12	11 … 0
0011	Addressing	Condition	Boolean address	Instructions type + serialize (9 instructions)	Count	Exec Address

Conditional_Execute_End						
47 … 44	43	42	41 … 34	33…16	15 …12	11 … 0
0100	Addressing	Condition	Boolean address	Instructions type + serialize (9 instructions)	Count	Exec Address

If the specified Boolean (8 bits can address 256 Booleans) meets the specified condition then execute the specified instructions (up to 9 instructions). If the condition is not met, we go on to the next control flow instruction. If Conditional_Execute_End and the condition is met, this is the last execution block of the shader program.

Conditional_Execute_Predicates							
47 … 44	43	42	41 … 36	35 … 34	33…16	15…12	11 … 0
0101	Addressing	Condition	RESERVED	Predicate vector	Instructions type + serialize (9 instructions)	Count	Exec Address

Conditional_Execute_Predicates_End							
47 … 44	43	42	41 … 36	35 … 34	33…16	15…12	11 … 0
0110	Addressing	Condition	RESERVED	Predicate vector	Instructions type + serialize (9 instructions)	Count	Exec Address

Check the AND/OR of all current predicate bits. If AND/OR matches the condition execute the specified number of instructions. We need to AND/OR this with the kill mask in order not to consider the pixels that aren't valid. If the

condition is not met, we go on to the next control flow instruction. If Conditional_Execute_Predicates_End and the condition is met, this is the last execution block of the shader program.

Conditional_Execute_Predicates_No_Stall							
47 … 44	43	42	41 … 36	35 … 34	33…16	15…12	11 … 0
1101	Addressing	Condition	RESERVED	Predicate vector	Instructions type + serialize (9 instructions)	Count	Exec Address

Conditional_Execute_Predicates_No_Stall_End							
47 … 44	43	42	41 … 36	35 … 34	33…16	15…12	11 … 0
1110	Addressing	Condition	RESERVED	Predicate vector	Instructions type + serialize (9 instructions)	Count	Exec Address

Same as Conditionnal_Execute_Predicates but the SQ is not going to wait for the predicate vector to be updated. You can only set this in the compiler if you know that the predicate set is only a refinement of the current one (like a nested if) because the optimization would still work.

Loop_Start					
47 … 44	43	42 … 21	20 … 16	15…12	11 … 0
0111	Addressing	RESERVED	loop ID	RESERVED	Jump address

Loop Start. Compares the loop iterator with the end value. If loop condition not met jump to the address. Forward jump only. Also computes the index value. The loop id must match between the start to end, and also indicates which control flow constants should be used with the loop.

Loop_End						
47 … 44	43	42 … 24	23… 21	20 … 16	15…12	11 … 0
1000	Addressing	RESERVED	Predicate break	loop ID	RESERVED	start address

Loop end. Increments the counter by one, compares the loop count with the end value. If loop condition met, continue, else, jump BACK to the start of the loop. If predicate break != 0, then compares predicate vector n (specified by predicate break number). If all bits cleared then break the loop.

The way this is described does not prevent nested loops, and the inclusion of the loop id make this easy to do.

Conditionnal_Call						
47 … 44	43	42	41 … 34	33 … 13	12	11 … 0
1001	Addressing	Condition	Boolean address	RESERVED	Force Call	Jump address

If the condition is met, jumps to the specified address and pushes the control flow program counter on the stack. If force call is set the condition is ignored and the call is made always.

Return		
47 … 44	43	42 … 0
1010	Addressing	RESERVED

Pops the topmost address from the stack and jumps to that address. If nothing is on the stack, the program will just continue to the next instruction.

Conditionnal_Jump							
47 … 44	43	42	41… 34	33	32 … 13	12	11 … 0
1011	Addressing	Condition	Boolean address	FW only	RESERVED	Force Jump	Jump address

If force jump is set the condition is ignored and the jump is made always. If FW only is set then only forward jumps are allowed.

Allocate				
47 … 44	43	42…41	40 … 4	3 …0
1100	Debug	Buffer Select	RESERVED	Allocation size

Buffer Select takes a value of the following:
01 – position export (ordered export)
10 – parameter cache or pixel export (ordered export)
11 – pass thru (out of order exports).

Buffer Size takes a value of the following:
00 – 1 buffer
01 – 2 buffers
…
15 – 16 buffers

If debug is set this is a debug alloc (ignore if debug DB_ON register is set to off).

6.3 Implementation

The envisioned implementation has a buffer that maintains the state of each thread. A thread lives in a given location in the buffer during its entire life, but the buffer has FIFO qualities in that threads leave in the order that they enter. Actually two buffers are maintained -- one for Vertices and one for Pixels. The intended implementation would allow for:

16 entries for vertices
48 entries for pixels.

From each buffer, arbitration logic attempts to select 1 thread for the texture unit and 1 (interleaved) thread for the ALU unit. Once a thread is selected it is read out of the buffer, marked as invalid, and submitted to appropriate execution unit. It is returned to the buffer (at the same place) with its status updated once all possible sequential instructions have been executed. A switch from ALU to TEX or visa-versa or a Serialize_Execution modifier forces the thread to be returned to the buffer.

Each entry in the buffer will be stored across two physical pieces of memory - most bits will be stored in a 1 read port device. Only bits needed for thread arbitration will be stored in a highly multi-ported structure. The bits kept in the 1 read port device will be termed 'state'. The bits kept in the multi-read ported device will be termed 'status'.

'State Bits' needed include:

1. Control Flow Instruction Pointer (13 bits),
2. Execution Count Marker 4 bits),
3. Loop Iterators (4x9 bits),
4. Call return pointers (4x12 bits),
5. Predicate Bits (64 bits),
6. Export ID (1 bit),
7. Parameter Cache base Ptr (7 bits),
8. GPR Base Ptr (8 bits),
9. Context Ptr (3 bits).
10. LOD corrections (6x16 bits)
11. Valid bits (64 bits)

Absent from this list are 'Index' pointers. These are costly enough that I'm presuming that they are instead stored in the GPRs. The first seven fields above (Control Flow Ptr, Execution Count, Loop Counts, call return ptrs, Predicate bits, PC base ptr and export ID) are updated every time the thread is returned to the buffer based on how much progress has been mode on thread execution. GPR Base Ptr, Context Ptr and LOD corrections are unchanged throughout execution of the thread.

'Status Bits' needed include:

- Valid Thread
- Texture/ALU engine needed
- Texture Reads are outstanding
- Waiting on Texture Read to Complete
- Allocation Wait (2 bits)
- 00 – No allocation needed
- 01 – Position export allocation needed (ordered export)
- 10 – Parameter or pixel export needed (ordered export)
- 11 – pass thru (out of order export)
- Allocation Size (4 bits)
- Position Allocated
- First thread of a new context
- Event thread (NULL thread that needs to trickle down the pipe)
- Last (1 bit)
- Pulse SX (1 bit)

All of the above fields from all of the entries go into the arbitration circuitry. The arbitration circuitry will select a winner for both the Texture Engine and for the ALU engine. There are actually two sets of arbitration -- one for pixels and one for vertices. A final selection is then done between the two. But the rest of this implementation summary only considers the 'first' level selection which is similar for both pixels and vertices.

Texture arbitration requires no allocation or ordering so it is purely based on selecting the 'oldest' thread that requires the Texture Engine.

ALU arbitration is a little more complicated. First, only threads where either of Texture_Reads_outstanding or Waiting_on_Texture_Read_to_Complete are '0' are considered. Then if Allocation_Wait is active, these threads are further filtered based on whether space is available. If the allocation is position allocation, then the thread is only considered if all 'older' threads have already done their position allocation (position allocated bits set). If the allocation is parameter or pixel allocation, then the thread is only considered if it is the oldest thread. Also a thread is not considered if it is a parameter or pixel or position allocation, has its First_thread_of_a_new_context bit set and would cause ALU interleaving with another thread performing the same parameter or pixel or position allocation. Finally the 'oldest' of the threads that pass through the above filters is selected. If the thread needed to allocate, then at this time the allocation is done, based on Allocation_Size. If a thread has its “last” bit set, then it is also removed from the buffer, never to return.

If I now redefine 'clauses' to mean 'how many times the thread is removed from the thread buffer for the purpose of exection by either the ALU or Texture engine', then the minimum number of clauses needed is 2 -- one to perform the allocation for exports (execution automatically halts after an 'Alloc' instruction) (but doesn't performs the actual allocation) and one for the actual ALU/export instructions. As the 'Alloc' instruction could be part of a texture clause (presumably the final instruction in such a clause), a thread could still execute in this minimal number of 2 clauses, even if it involved texture fetching.

The Texture_Reads_Outstanding bit must be updated by the sequencer, based on keeping track of how many Texture Clauses have been executed by a given thread that have not yet had there data returned. Any number above 0 results in this bit being set. We could consider forcing synchronization such that two texture clauses for a given thread may not be outstanding at any time (that would be my preference for simplicity reasons and because it would require only very little change in the texture pipe interface). This would allow the sequencer to set the bit on execution of the texture clause, and allow the texture unit to return a pointer to the thread buffer on completion that clears the bit.

6.4 Data dependant predicate instructions

Data dependant conditionals will be supported in the R400. The only way we plan to support those is by supporting three vector/scalar predicate operations of the form:

PRED_SETE_# - similar to SETE except that the result is 'exported' to the sequencer.
PRED_SETNE_# - similar to SETNE except that the result is 'exported' to the sequencer.
PRED_SETGT_# - similar to SETGT except that the result is 'exported' to the sequencer
PRED_SETGTE_# - similar to SETGTE except that the result is 'exported' to the sequencer

For the scalar operations only we will also support the two following instructions:
PRED_SETE0_# – SETE0
PRED_SETE1_# – SETE1

The export is a single bit - 1 or 0 that is sent using the same data path as the MOVA instruction. The sequencer will maintain 4 sets of 64 bit predicate vectors (in fact 8 sets because we interleave two programs but only 4 will be exposed) and use it to control the write masking. This predicate is not maintained across clause boundaries. The # sign is used to specify which predicate set you want to use 0 thru 3.

Then we have two conditional execute bits. The first bit is a conditional execute “on” bit and the second bit tells us if we execute on 1 or 0. For example, the instruction:

P0_ADD_# R0,R1,R2

Is only going to write the result of the ADD into those GPRs whose predicate bit is 0. Alternatively, P1_ADD_# would only write the results to the GPRs whose predicate bit is set. The use of the P0 or P1 without precharging the sequencer with a PRED instruction is undefined.

{Issue: do we have to have a NOP between PRED and the first instruction that uses a predicate?}

6.5 HW Detection of PV,PS

Because of the control program, the compiler cannot detect statically dependant instructions. In the case of non-masked writes and subsequent reads the sequencer will insert uses of PV,PS as needed. This will be done by comparing the read address and the write address of consecutive instructions. For masked writes, the sequencer will insert NOPs wherever there is a dependant read/write.

The sequencer will also have to insert NOPs between PRED_SET and MOVA instructions and their uses.

6.6 Register file indexing

Because we can have loops in fetch clause, we need to be able to index into the register file in order to retrieve the data created in a fetch clause loop and use it into an ALU clause. The instruction will include the base address for register indexing and the instruction will contain these controls:

Bit7	Bit 6	
0	0	'absolute register'
0	1	'relative register'
1	0	'previous vector'
1	1	'previous scalar'

In the case of an absolute register we just take the address as is. In the case of a relative register read we take the base address and we add to it the loop_index and this becomes our new address that we give to the shader pipe.

The sequencer is going to keep a loop index computed as such:

Index = Loop_iterator*Loop_step + Loop_start.

We loop until loop_iterator = loop_count. Loop_step is a signed value [-128…127]. The computed index value is a 10 bit counter that is also signed. Its real range is [-256,256]. The tenth bit is only there so that we can provide an out of range value to the “indexing logic” so that it knows when the provided index is out of range and thus can make the necessary arrangements.

6.7 Debugging the Shaders

In order to be able to debug the pixel/vertex shaders efficiently, we provide 2 methods.

6.7.1 *Method 1: Debugging registers*

Current plans are to expose 2 debugging, or error notification, registers:
1. address register where the first error occurred
2. count of the number of errors

The sequencer will detect the following groups of errors:
- count overflow
- constant indexing overflow
- register indexing overflow

Compiler recognizable errors:
- jump errors
    relative jump address > size of the control flow program
- call stack
    call with stack full
    return with stack empty

A jump error will always cause the program to break. In this case, a break means that a clause will halt execution, but allowing further clauses to be executed.

With all the other errors, program can continue to run, potentially to worst-case limits. The program will only break if the DB_PROB_BREAK register is set.

If indexing outside of the constant or the register range, causing an overflow error, the hardware is specified to return the value with an index of 0. This could be exploited to generate error tokens, by reserving and initializing the 0th register (or constant) for errors.

{ISSUE : Interrupt to the driver or not?}

6.7.2 *Method 2: Exporting the values in the GPRs*

1) The sequencer will have a debug active, count register and an address register for this mode.

Under the normal mode execution follows the normal course.

Under the debug mode it is assumed that the program is always exporting n debug vectors and that all other exports to the SX block (position, color, z, ect) will been turned off (changed into NOPs) by the sequencer (even if they occur before the address stated by the ADDR debug register).

7. Pixel Kill Mask

A vector of 64 bits is kept by the sequencer per group of pixels/vertices. Its purpose is to optimize the texture fetch requests and allow the shader pipe to kill pixels using the following instructions:

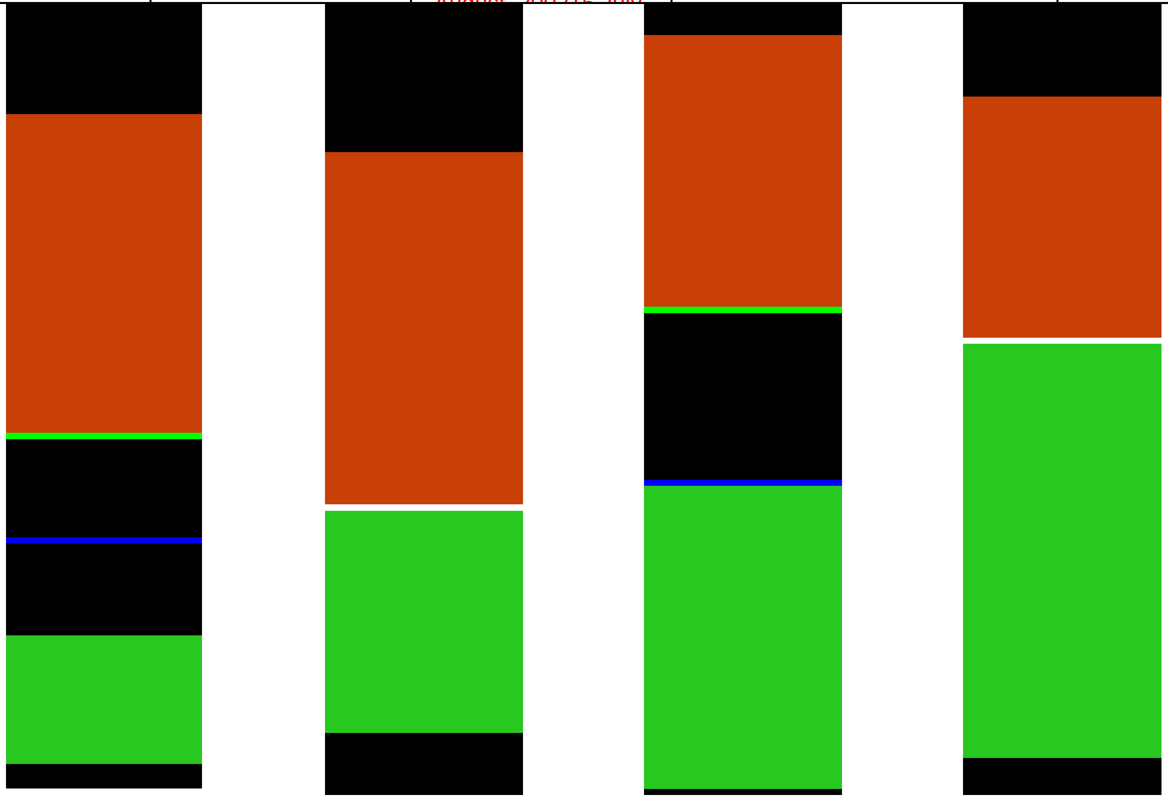
MASK_SETE
MASK_SETNE
MASK_SETGT
MASK_SETGTE

8. Multipass vertex shaders (HOS)

Multipass vertex shaders are able to export from the 6 last clauses but to memory ONLY.

9. Register file allocation

The register file allocation for vertices and pixels can either be static or dynamic. In both cases, the register file in managed using two round robins (one for pixels and one for vertices). In the dynamic case the boundary between pixels and vertices is allowed to move, in the static case it is fixed to 128-VERTEX_REG_SIZE for vertices and PIXEL_REG_SIZE for pixels.



Above is an example of how the algorithm works. Vertices come in from top to bottom; pixels come in from bottom to top. Vertices are in orange and pixels in green. The blue line is the tail of the vertices and the green line is the tail of the pixels. Thus anything between the two lines is shared. When pixels meets vertices the line turns white and the boundary is static until both vertices and pixels share the same “unallocated bubble”. Then the boundary is allowed to move again. The numbering of the GPRs starts from the bottom of the picture at index 0 and goes up to the top at index 127.

10. Fetch Arbitration

The fetch arbitration logic chooses one of the n potentially pending fetch clauses to be executed. The choice is made by looking at the Vs and Ps reservation stations and picking the first one ready to execute. Once chosen, the clause state machine will send one 2x2 fetch per clock (or 4 fetches in one clock every 4 clocks) until all the fetch instructions of the clause are sent. This means that there cannot be any dependencies between two fetches of the same clause.

The arbitrator will not wait for the fetches to return prior to selecting another clause for execution. The fetch pipe will be able to handle up to X(?) in flight fetches and thus there can be a fair number of active clauses waiting for their fetch return data.

11. ALU Arbitration

ALU arbitration proceeds in almost the same way than fetch arbitration. The ALU arbitration logic chooses one of the n potentially pending ALU clauses to be executed. The choice is made by looking at the Vs and Ps reservation stations and picking the first one ready to execute. There are two ALU arbiters, one for the even clocks and one for the odd clocks. For example, here is the sequencing of two interleaved ALU clauses (E and O stands for Even and Odd sets of 4 clocks):

Einst0 Oinst0 Einst1 Oinst1 Einst2 Oinst2 Einst0 Oinst3 Einst1 Oinst4 Einst2 Oinst0…

Proceeding this way hides the latency of 8 clocks of the ALUs. Also note that the interleaving also occurs across clause boundaries.

12. Handling Stalls

When the output file is full, the sequencer prevents the ALU arbitration logic from selecting the last clause (this way nothing can exit the shader pipe until there is place in the output file. If the packet is a vertex packet and the position buffer is full (POS_FULL) then the sequencer also prevents a thread from entering an exporting clause. The sequencer will set the OUT_FILE_FULL signal n clocks before the output file is actually full and thus the ALU arbiter will be able read this signal and act accordingly by not preventing exporting clauses to proceed.

13. Content of the reservation station FIFOs

The reservation FIFOs contain the state of the vector of pixels and vertices. We have two sets of those: one for pixels, and one for vertices. They contain 3 bits of Render State 7 bits for the base address of the GPRs, some bits for LOD correction and coverage mask information in order to fetch fetch for only valid pixels, the quad address.

14. The Output File

The output file is where pixels are put before they go to the RBs. The write BW to this store is 256 bits/clock. Just before this output file are staging registers with write BW 512 bits/clock and read BW 256 bits/clock. The staging registers are 4x128 (and there are 16 of those on the whole chip).

15. IJ Format

The IJ information sent by the PA is of this format on a per quad basis:

We have a vector of IJ's (one IJ per pixel at the centroid of the fragment or at the center of the pixel depending on the mode bit). All pixel's parameters are always interpolated at full 20x24 mantissa precision.

$$P0 = A + I(0)*(B-A) + J(0)*(C-A)$$
$$P1 = A + I(1)*(B-A) + J(1)*(C-A)$$
$$P2 = A + I(2)*(B-A) + J(2)*(C-A)$$
$$P3 = A + I(3)*(B-A) + J(3)*(C-A)$$

P0	P1
P2	P3

Multiples (Full Precision): 8
Subtracts 19x24 (Parameters): 2
Adds: 8

FORMAT OF P's IJ : Mantissa 20 Exp 4 for I + Sign
Mantissa 20 Exp 4 for J + Sign

Total number of bits : 20*8 + 4*8 + 4*2 = 200.

All numbers are kept using the un-normalized floating point convention: if exponent is different than 0 the number is normalized if not, then the number is un-normalized. The maximum range for the IJs (Full precision) is +/- 1024.

15.1 Interpolation of constant attributes

Because of the floating point imprecision, we need to take special provisions if all the interpolated terms are the same or if two of the terms are the same.

16. Staging Registers

In order for the reuse of the vertices to be 14, the sequencer will have to re-order the data sent IN ORDER by the VGT for it to be aligned with the parameter cache memory arrangement. Given the following group of vertices sent by the VGT:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 || 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 || 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 || 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63

The sequencer will re-arrange them in this fashion:

0 1 2 3 16 17 18 19 32 33 34 35 48 49 50 51 || 4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55 || 8 9 10 11 24 25 26 27 40 41 42 43 56 57 58 59 || 12 13 14 15 28 29 30 31 44 45 46 47 60 61 62 63

The || markers show the SP divisions. In the event a shader pipe is broken, the SQ ~~VGT will send~~is responsible to insert padding to account for the missing pipe. For example, if SP1 is broken, vertices 4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55 will ~~still be~~ not be sent by the VGT to the SQ ~~**BUT**~~ **AND** the SQ is responsible to “jump” over these vertices in order for no valid vertices to be sent to an invalid SP ~~will not be processed by the SP and thus should be considered invalid (by the SU and VGT)~~.

The most straightforward, *non-compressed* interface method would be to convert, in the VGT, the data to 32-bit floating point prior to transmission to the VSISRs. In this scenario, the data would be transmitted to (and stored in) the VSISRs in full 32-bit floating point. This method requires three 24-bit fixed-to-float converters in the VGT. Unfortunately, it also requires and additional 3,072 bits of storage across the VSISRs. This interface is illustrated in Figure 11~~Figure 11Figure 11~~. The area of the fixed-to-float converters and the VSISRs for this method is roughly estimated as 0.759sqmm using the R300 process. The gate count estimate is shown in Figure 10~~Figure 10Figure 10~~.

Basis for 8-deep Latch Memory (from R300)			
8x24-bit	11631 μ²		60.57813 μ² per bit
Area of 96x8-deep Latch Memory	46524 μ²		
Area of 24-bit Fix-to-float Converter	4712 μ² per converter		
Method 1	Block	Quantity	Area
	F2F	3	14136
	8x96 Latch	16	744384
			758520 μ²

Figure 10:Area Estimate for VGT to Shader Interface

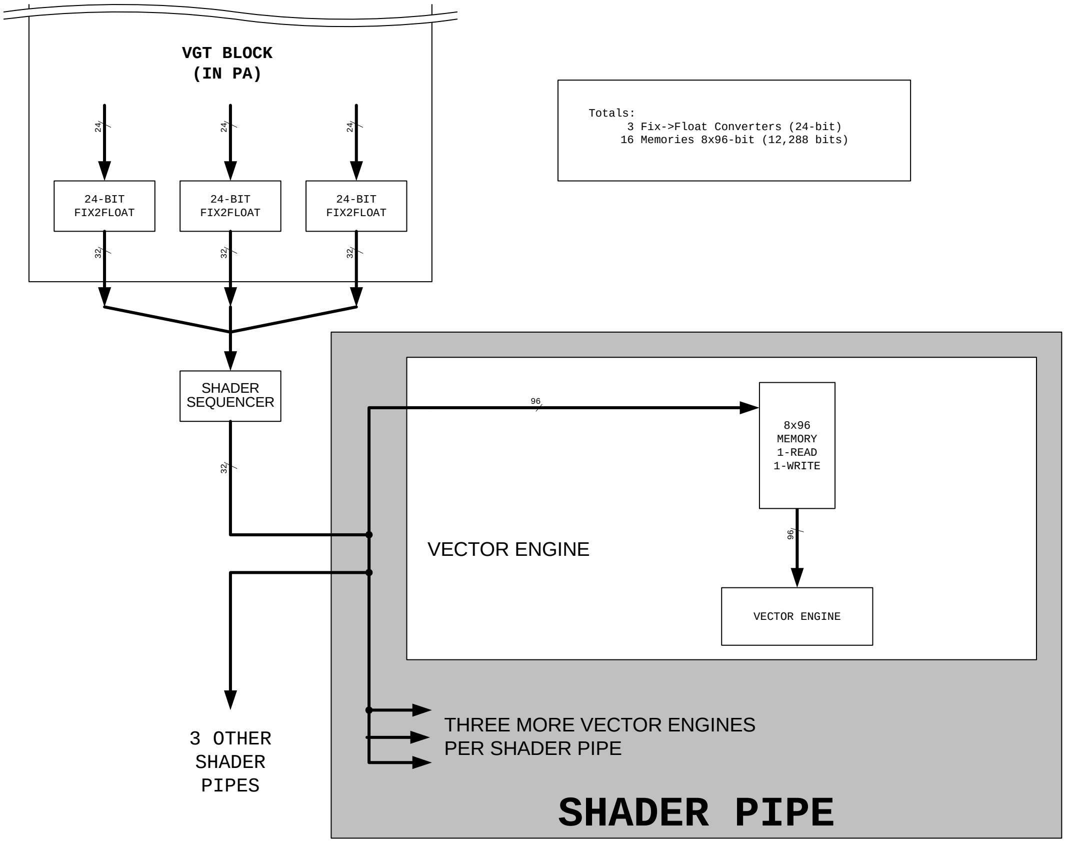


Figure 11:VGT to Shader Interface

17. The parameter cache

The parameter cache is where the vertex shaders export their data. It consists of 16 128x128 memories (1R/1W). The reuse engine will make it so that all vertexes of a given primitive will hit different memories. The allocation method for these memories is a simple round robin. The parameter cache pointers are mapped in the following way: 4MSBs are the memory number and the 7 LSBs are the address within this memory.

MEMORY NUMBER 4 bits	ADDRESS 7 bits

The PA generates the parameter cache addresses as the positions come from the SQ. All it needs to do is keep a Current_Location pointer (7 bits only) and as the positions comes increment the memory number. When the memory number field wraps around, the PA increments the Current_Location by VS_EXPORT_COUNT (a snooped register from the SQ). As an example, say the memories are all empty to begin with and the vertex shader is exporting 8 parameters per vertex (VS_EXPORT_COUNT = 8). The first position received is going to have the PC address 00000000000 the second one 00010000000, third one 00100000000 and so on up to 11110000000. Then the next position received (the 17th) is going to have the address 00000001000, the 18th 00010001000, the 19th 00100001000 and so on. The Current_location is NEVER reset BUT on chip resets. The only thing to be careful about is that if the SX doesn't send you a full group of positions (<64) then you need to fill the address space so that the next group starts correctly aligned (for example if you receive only 33 positions then you need to add 2*VS_EXPORT_COUNT to Current_Location and reset the memory count to 0 before the next vector begins).

17.1 Export restrictions

17.1.1 *Pixel exports:*

Pixels can export 1,2,3 or 4 color buffers to the SX(+z). The exports will be done in order. The PRED_OPTIMIZE function has to be turned of if the exports are done using interleaved predicated instructions. The exports will always be ordered to the SX.

17.1.2 *Vertex exports:*

Position or parameter caches can be exported in any order in the shader program. It is always better to export posistion as soon as possible. Position has to be exported in a single export block (no texture instructions can be placed between the exports). Parameter cache exports can be done in any order with texture instructions interleaved. The PRED_OPTIMIZE function has to be turned of if the exports are done using interleaved predicated instructions to the Parameter cache (see Arbitration restrictions for details). The exports will always be allocated in order to the SX.

17.1.3 *Pass thru exports:*

Pass thru exports have to be done in groups of the form:

```
Alloc 4 (8 or 12)
Execute ALU(ADDR) ALU(DATA) ALU(DATA) ALU(DATA)…
```

They cannot have texture instructions interleaved in the export block. These exports **are not guaranteed to be ordered**.

Also, when doing a pass thru export, Position MUST be exported AFTER all pass thru exports. This position export is used to synchronize the chip when doing a transition from pass thru shader to regular shader and vice versa.

17.2 Arbitration restrictions

Here are the Sequencer arbitration restrictions:

1) Cannot execute a serialized thread if the corresponding texture pending bit is set
2) Cannot allocate position if any older thread has not allocated position
3) If last thread is marked as not valid AND marked as last and we are about to execute the second to oldest thread also marked last then:
    a. Both threads must be from the same context (cannot allow a first thread)
    b. Must turn off the predicate optimization for the second thread
4) Cannot execute a texture clause if texture reads are pending
5) Cannot execute last if texture pending (even if not serial)

18. Export Types

The export type (or the location where the data should be put) is specified using the destination address field in the ALU instruction. Here is a list of all possible export modes:

18.1 Vertex Shading

0:15	- 16 parameter cache
16:31	- Empty (Reserved?)
32	- Export Address
33:40	- 8 vertex exports to the frame buffer and index
41:47	- Empty
48:55	- 8 debug export (interpret as normal vertex export)
60	- export addressing mode
61	- Empty
62	- position

63	- sprite size export that goes with position export (point_h,point_w,edgeflag,misc)

18.2 Pixel Shading

0	- Color for buffer 0 (primary)
1	- Color for buffer 1
2	- Color for buffer 2
3	- Color for buffer 3
4:7	- Empty
8	- Buffer 0 Color/Fog (primary)
9	- Buffer 1 Color/Fog
10	- Buffer 2 Color/Fog
11	- Buffer 3 Color/Fog
12:15	- Empty
16:31	- Empty (Reserved?)
32	- Export Address
33:40	- 8 exports for multipass pixel shaders.
41:47	- Empty
48:55	- 8 debug exports (interpret as normal pixel export)
60	- export addressing mode
61:62	- Empty
63	- Z for primary buffer (Z exported to 'alpha' component)

19. Special Interpolation modes

19.1 Real time commands

We are unable to use the parameter memory since there is no way for a command stream to write into it. Instead we need to add three 16x128 memories (one for each of three vertices x 16 interpolants). These will be mapped onto the register bus and written by type 0 packets, and output to the the parameter busses (the sequencer and/or PA need to be able to address the reatime parameter memory as well as the regular parameter store. For higher performance we should be able able to view them as two banks of 16 and do double buffering allowing one to be loaded, while the other is rasterized with. Most overlay shaders will need 2 or 4 scalar coordinates, one option might be to restrict the memory to 16x64 or 32x64 allowing only two interpolated scalars per cycle, the only problem I see with this is, if we view support for 16 vector-4 interpolants important (true only if we map Microsoft's high priority stream to the realtime stream), then the PA/sequencer need to support a realtime-specific mode where we need to address 32 vectors of parameters instead of 16. This mode is triggered by the primitive type: REAL TIME. The actual memories are in the in the SX blocks. The parameter data memories are hooked on the RBBM bus and are loaded by the CP using register mapped memory.

19.2 Sprites/ XY screen coordinates/ FB information

~~When working with sprites, one may want to overwrite the parameter 0 with SC generated data. Also,~~ XY screen coordinates may be needed in the shader program. This functionality is controlled by the param_gen_I0 register (in SQ) in conjunction with the SND_XY register (in SC) and the param_gen_pos. Also it is possible to send the faceness information (for OGL front/back special operations) to the shader using the same control register. Here is a list of all the modes and how they interact together:

The Data is going to be written in the register specified by the param_gen_pos register.

~~Gen_st is a bit taken from the interface between the SC and the SQ. This is the MSB of the primitive type. If the bit is set, it means we are dealing with Point AA, Line AA or sprite and in this case the vertex values are going to generated between 0 and 1.~~

Param_Gen_I0 disable, snd_xy disable~~, no gen_st – I0~~ = No modification
~~Param_Gen_I0 disable, snd_xy disable, gen_st – I0 = No modification~~
Param_Gen_I0 disable, snd_xy enable~~, no gen_st – I0~~ = No modification

~~Param_Gen_I0 disable, snd_xy enable, gen_st – I0 = No modification~~
Param_Gen_I0 enable, snd_xy disable~~, no gen_st – I0~~ = Sign(faceness)garbage,(Sign Point)~~-~~garbage,Sign(Line) ~~garbage, facenesss~~, t
~~Param_Gen_I0 enable, snd_xy disable, gen_st – I0 = garbage, garbage, s, t~~
Param_Gen_I0 enable, snd_xy enable~~, no gen_st – I0~~ = Sign(faceness)screenX,(Sign Point)screenY,Sign(Line)s, t

In other words,

The generated vector is (X in RED, Y in GREEN, S in BLUE and T in ALPHA):
X,Y,S,T
These values are always supposed to be positive and any shader use of them should use the ABS function (as their sign bits will now be used for flags).
SignX = BackFacing
SignY = Point Primitive
SignS = Line Primitive
SignT = currently unused as a flag.

If !Point & !Line, then it is a Poly.

I would assume that one implementation which allows for generic texture lookup (using 3D maps) for poly stipple and AA for the driver would be

if(Y<0) {
        R = 0.0 (Point)
} else if (S < 0) {
        R = 1.0 (Line)
} else {
        R = 2.0 (Poly)
}~~screen x, screen y, garbage, faceness~~
~~Param_Gen_I0 enable, snd_xy enable, gen_st – I0 = screen x, screen y, s, t~~

19.3 Auto generated counters

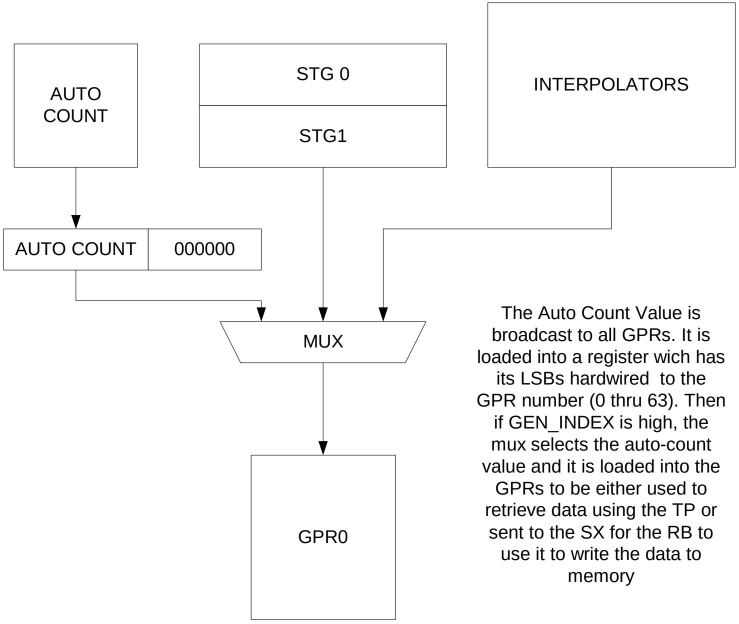
In the cases we are dealing with multipass shaders, the sequencer is going to generate a vector count to be able to both use this count to write the 1st pass data to memory and then use the count to retrieve the data on the 2nd pass. The count is always generated in the same way but it is passed to the shader in a slightly different way depending on the shader type (pixel or vertex). This is toggled on and off using the GEN_INDEX register. The sequencer is going to keep two counters, one for pixels and one for vertices. Every time a full vector of vertices or pixels is written to the GPRs the counter is incremented. Every time a state change is detected, the corresponding counter is reset. While there is only one count broadcast to the GPRs, the LSB are hardwired to specific values making the index different for all elements in the vector.

19.3.1 *Vertex shaders*

In the case of vertex shaders, if GEN_INDEX is set, the data will be put into the x field of the third register (it means that the compiler must allocate 3 GPRs in all multipass vertex shader modes).

19.3.2 *Pixel shaders*

In the case of pixel shaders, if GEN_INDEX is set~~ and Param_Gen_I0 is enabled~~, the data will be put in the x field of the ~~2nd~~ ~~-~~param_gen_pos+1 register~~ (R1.x), else if GEN_INDEX is set the data will be put into the x field of the 1st register (R0.x)~~.



The Auto Count Value is broadcast to all GPRs. It is loaded into a register wich has its LSBs hardwired to the GPR number (0 thru 63). Then if GEN_INDEX is high, the mux selects the auto-count value and it is loaded into the GPRs to be either used to retrieve data using the TP or sent to the SX for the RB to use it to write the data to memory

Figure 12: GPR input mux Control

20. State management

Every clock, the sequencer will report to the CP the oldest states still in the pipe. These are the states of the programs as they enter the last ALU clause.

20.1 Parameter cache synchronization

In order for the sequencer not to begin a group of pixels before the associated group of vertices has finished, the sequencer will keep a 6 bit count per state (for a total of 8 counters). These counters are initialized to 0 and every time a vertex shader exports its data TO THE PARAMETER CACHE, the corresponding pointer is incremented. When the SC sends a new vector of pixels with the SC_SQ_new_vector bit asserted, the sequencer will first check if the count is greater than 0 before accepting the transmission (it will in fact accept the transmission but then lower its ready to receive). Then the sequencer waits for the count to go to one and decrements it. The sequencer can then issue the group of pixels to the interpolators. Every time the state changes, the new state counter is initialized to 0.

21. XY Address imports

The SC will be able to send the XY addresses to the GPRs. It does so by interleaving the writes of the IJs (to the IJ buffer) with XY writes (to the XY buffer). Then when writing the data to the GPRs, the sequencer is going to interpolate the IJ data or pass the XY data thru a Fix→float converter and expander and write the converted values to the GPRs. The Xys are currently SCREEN SPACE COORDINATES. The values in the XY buffers will wrap. See section 19.2 for details on how to control the interpolation in this mode.

21.1 Vertex indexes imports

In order to import vertex indexes, we have 16 8x96 staging registers. These are loaded one line at a time by the VGT block (96 bits). They are loaded in floating point format and can be transferred in 4 or 8 clocks to the GPRs.

22. Registers

Please see the auto-generated web pages for register definitions.

23. Interfaces

23.1 External Interfaces

Whenever an x is used, it means that the bus is broadcast to all units of the same name. For example, if a bus is named SQ→SPx it means that SQ is going to broadcast the same information to all SP instances.

23.2 SC to SP Interfaces

23.2.1 *SC_SP#*

There is one of these interfaces at front of each of the SP (buffer to stage pixel interpolators). This interface transmits the I,J data for pixel interpolation. For the entire system, two quads per clock are transferred to the 4 SPs, so each of these 4 interfaces transmits one half of a quad per clock. The interface below describes a half of a quad worth of data.
The actual data which is transferred per quad is
Ref Pix I => S4.20 Floating Point I value *4
Ref Pix J => S4.20 Floating Point J value *4

This equates to a total of 200 bits which transferred over 2 clocks
and therefor needs an interface 100 bits wide

Additionally, X,Y data (12-bit unsigned fixed) is conditionally sent across this data bus over the same wires in an additional clock. The X,Y data is sent on the lower 24 bits of the data bus with faceness in the msb.
Transfers across these interfaces are synchronized with the SC_SQ IJ Control Bus transfers.

The data transfer across each of these busses is controlled by a IJ_BUF_INUSE_COUNT in the SC. Each time the SC has sent a pixel vector's worth of data to the SPs, he will increment the IJ_BUF_INUSE_COUNT count. Prior to sending the next pixel vectors data, he will check to make sure the count is less than MAX_BUFER_MINUS_2, if not the SC will stall until the SQ returns a pipelined pulse to decrement the count when he has scheduled a buffer free. Note: We could/may optimize for the case of only sending only IJ to use all the buffers to pre-load more. Currently it is planned for the SP to hold 2 double buffers of I,J data and two buffers of X,Y data, so if either X,Y or Centers and Centroids are on, then the SC can send two Buffers.

In at least the initial version, the SC shall send 16 quads per pixel vector even if the vector is not full. This will increment buffer write address pointers correctly all the time. (We may revisit this for both the SX,SP,SQ and add a EndOfVector signal on all interfaces to quit early. We opted for the simple mode first with a belief that only the end of packet and multiple new vector signals should cause a partial vector and that this would not really be significant performance hit.)

Name	Bits	Description
SC_SP#_data	100	IJ information sent over 2 clocks (or X,Y in 24 LSBs with faceness in upper bit) **Type 0 or 1**, First clock I, second clk J Field ULC URC LLC LRC Bits [63:39] [38:26] [25:13] [12:0] Format SE4M20 SE4M20 SE4M20 SE4M20 **Type 2** Field Face X Y Bits [63] [23:12] [11:0] Format Bit Unsigned Unsigned
SC_SP#_valid	1	Valid
SC_SP#_last_quad_data	1	This bit will be set on the last transfer of data per quad.
SC_SP#_type	2	0 -> Indicates centroids 1 -> Indicates centers 2 -> Indicates X,Y Data and faceness on data bus The SC shall look at state data to determine how many types to send for the

		interpolation process.

The # is included for clarity in the spec and will be replaced with a prefix of u#_ in the verilog module statement for the SC and the SP block will have neither because the instantiation will insert the prefix.

23.2.2 *SC_SQ*

This is the control information sent to the sequencer in order to synchronize and control the interpolation and/or loading data into the GPRs needed to execute a shader program on the sent pixels. This data will be sent over two clocks per transfer with 1 to 16 transfers. Therefore the bus (approx 94 bits) could be folded in half to approx 49 bits.

Name	**Bits**	**Description**
SC_SQ_data	46	Control Data sent to the SQ 1 clk transfers Event – valid data consist of event_id and state_id. Instruct SQ to post an event vector to send state id and event_id through request fifo and onto the reservation stations making sure state id and/or event_id gets back to the CP. Events only follow end of packets so no pixel vectors will be in progress. Empty Quad Mask – Transfer Control data consisting of pc_dealloc or new_vector. Receipt of this is to transfer pc_dealloc or new_vector without any valid quad data. New vector will always be posted to request fifo and pc_dealloc will be attached to any pixel vector outstanding or posted in request fifo if no valid quad outstanding. 2 clk transfers Quad Data Valid – Sending quad data with or without new_vector or pc_dealloc. New vector will be posted to request fifo with or without a pixel vector and pc_dealloc will be posted with a pixel vector unless none is in progress. In this case the pc_dealloc will be posted in the request queue. Filler quads will be transferred with The Quad mask set but the pixel corresponding pixel mask set to zero.
SC_SQ_valid	1	SC sending valid data, 2nd clk could be all zeroes

SC_SQ_data – first clock and second clock transfers are shown in the table below.

Name	**BitField**	**Bits**	**Description**
1st Clock Transfer			
SC_SQ_event	0	1	This transfer is a 1 clock event vector Force quad_mask = new_vector=pc_dealloc=0
SC_SQ_event_id	[4:1]	4	This field identifies the event 0 => denotes an End Of State Event 1 => TBD

SC_SQ_pc_dealloc	[7:5]	3	Deallocation token for the Parameter Cache
SC_SQ_new_vector	8	1	The SQ must wait for Vertex shader done count > 0 and after dispatching the Pixel Vector the SQ will decrement the count.
SC_SQ_quad_mask	[12:9]	4	Quad Write mask left to right SP0 => SP3
SC_SQ_end_of_prim	13	1	End Of the primitive
SC_SQ_state_id	[16:14]	3	State/constant pointer (6*3+3)
SC_SQ_pix_mask	[32:17]	16	Valid bits for all pixels SP0=>SP3 (UL,UR,LL,LR)
SC_SQ_provok_vtx	[37:36]	2	Provoking vertex for flat shading
SC_SQ_pc_ptr0	[48:38]	11	Parameter Cache pointer for vertex 0
2nd Clock Transfer			
SC_SQ_pc_ptr1	[10:0]	11	Parameter Cache pointer for vertex 1
SC_SQ_pc_ptr2	[21:11]	11	Parameter Cache pointer for vertex 2
SC_SQ_lod_correct	[45:22]	24	LOD correction per quad (6 bits per quad)
SC_SQ_prim_type	[48:46]	3	Stippled line and Real time command need to load tex cords from alternate buffer 000: Sprite (point) 001: Line 010: Tri_rect 100: Realtime Sprite (point) 101: Realtime Line 110: Realtime Tri_rect

Name	Bits	Description
SQ_SC_free_buff	1	Pipelined bit that instructs SC to decrement count of buffers in use.
SQ_SC_dec_cntr_cnt	1	Pipelined bit that instructs SC to decrement count of new vector and/or event sent to prevent SC from overflowing SQ interpolator/Reservation request fifo.

The scan converter will submit a partial vector whenever:

1.) He gets a primitive marked with an end of packet signal.
2.) A current pixel vector is being assembled with at least one or more valid quads and the vector has been marked for deallocate when a primitive marked new_vector arrives. The Scan Converter will submit a partial vector (up to 16quads with zero pixel mask to fill out the vector) prior to submitting the new_vector marker\primitive.

(This will prevent a hang which can be demonstrated when all primitives in a packet three vectors are culled except for a one quad primitive that gets marked pc_dealloc (vertices maximum size). In this case two new_vectors are submitted and processed, but then one valid quad with the pc_dealloc creates a vector and then the new would wait for another vertex vector to be processed, but the one being waited for could never export until the pc_dealloc signal made it through and thus the hang.)

23.2.3 *SQ to SX(SP): Interpolator bus*

Name	Direction	Bits	Description
SQ_SPx_interp_flat_vtx	SQ→SPx	2	Provoking vertex for flat shading
SQ_SPx_interp_flat_gouraud	SQ→SPx	1	Flat or gouraud shading
SQ_SPx_interp_cyl_wrap	SQ→SPx	4	Wich channel needs to be cylindrical wrapped
SQ_SXx_pc_ptr0	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_pc_ptr1	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_pc_ptr2	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_rt_sel	SQ→SXx	1	Selects between RT and Normal data
SQ_SXx_pc_wr_en	SQ→SXx	1	Write enable for the PC memories
SQ_SXx_pc_wr_addr	SQ→SXx	7	Write address for the PCs
SQ_SXx_pc_channel_mask	SQ→SXx	4	Channel mask
SQ_SXx_pc_ptr_valid	SQ→SXx	1	Read pointers are valid.
SQ_SPx_interp_valid	SQ→SPx	1	Interpolation control valid

23.2.4 *SQ to SP: Staging Register Data*

This is a broadcast bus that sends the VSISR information to the staging registers of the shader pipes.

Name	Direction	Bits	Description
SQ_SPx_vsr_data	SQ→SPx	96	Pointers of indexes or HOS surface information
SQ_SPx_vsr_double	SQ→SPx	1	0: Normal 96 bits per vert 1: double 192 bits per vert
SQ_SP0_ vsr_valid	SQ→SP0	1	Data is valid
SQ_SP1_ vsr_ valid	SQ→SP1	1	Data is valid
SQ_SP2_ vsr_ valid	SQ→SP2	1	Data is valid
SQ_SP3_ vsr_ valid	SQ→SP3	1	Data is valid
SQ_SPx_vsr_read	SQ→SPx	1	Increment the read pointers

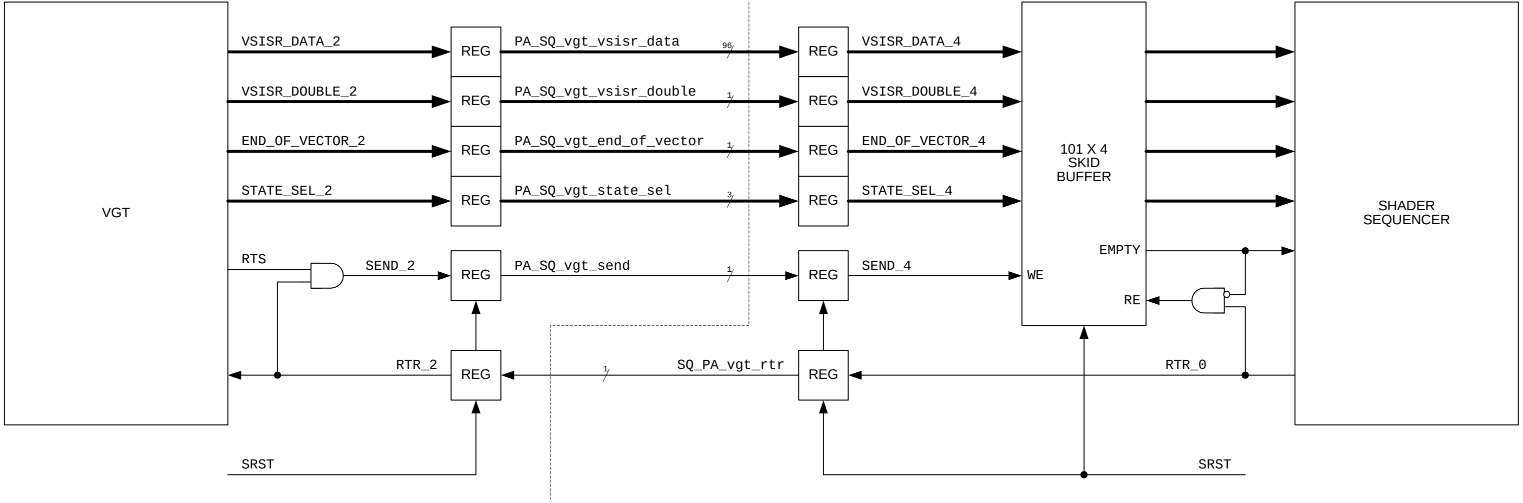
23.2.5 *VGT to SQ : Vertex interface*

23.2.5.1 Interface Signal Table

The area difference between the two methods is not sufficient to warrant complicating the interface or the state requirements of the VSISRs. **Therefore, the POR for this interface is that the VGT will transmit the data to the VSISRs (via the Shader Sequencer) in full, 32-bit floating-point format.** The VGT can transmit up to six 32-bit floating-point values to each VSISR where four or more values require two transmission clocks. The data bus is 96 bits wide.

Name	Bits	Description
VGT_SQ_vsisr_data	96	Pointers of indexes or HOS surface information
VGT_SQ_event	1	VGT is sending an event
VGT_SQ_vsisr_continued	1	0: Normal 96 bits per vert 1: double 192 bits per vert
VGT_SQ_end_of_vtx_vect	1	Indicates the last VSISR data set for the current process vector (for double vector data, "end_of_vector" is set on the first vector)
VGT_SQ_indx_valid	1	Vsisr data is valid
VGT_SQ_state	3	Render State (6*3+3 for constants). This signal is guaranteed to be correct when “VGT_SQ_vgt_end_of_vector” is high.
VGT_SQ_send	1	Data on the VGT_SQ is valid receive (see write-up for standard R400 SEND/RTR interface handshaking)
SQ_VGT_rtr	1	Ready to receive (see write-up for standard R400 SEND/RTR interface handshaking)

23.2.5.2 Interface Diagrams


VGT
VSISR_DATA_2
VSISR_DOUBLE_2
END_OF_VECTOR_2
STATE_SEL_2
RTS
SEND_2
RTR_2
SRST
REG
REG
REG
REG
REG
REG
PA_SQ_vgt_vsisr_data
96
PA_SQ_vgt_vsisr_double
1
PA_SQ_vgt_end_of_vector
1
PA_SQ_vgt_state_sel
3
PA_SQ_vgt_send
1
SQ_PA_vgt_rtr
1
REG
REG
REG
REG
REG
REG
VSISR_DATA_4
VSISR_DOUBLE_4
END_OF_VECTOR_4
STATE_SEL_4
SEND_4
101 X 4
SKID
BUFFER
WE
EMPTY
RE
RTR_0
SRST
SHADER
SEQUENCER

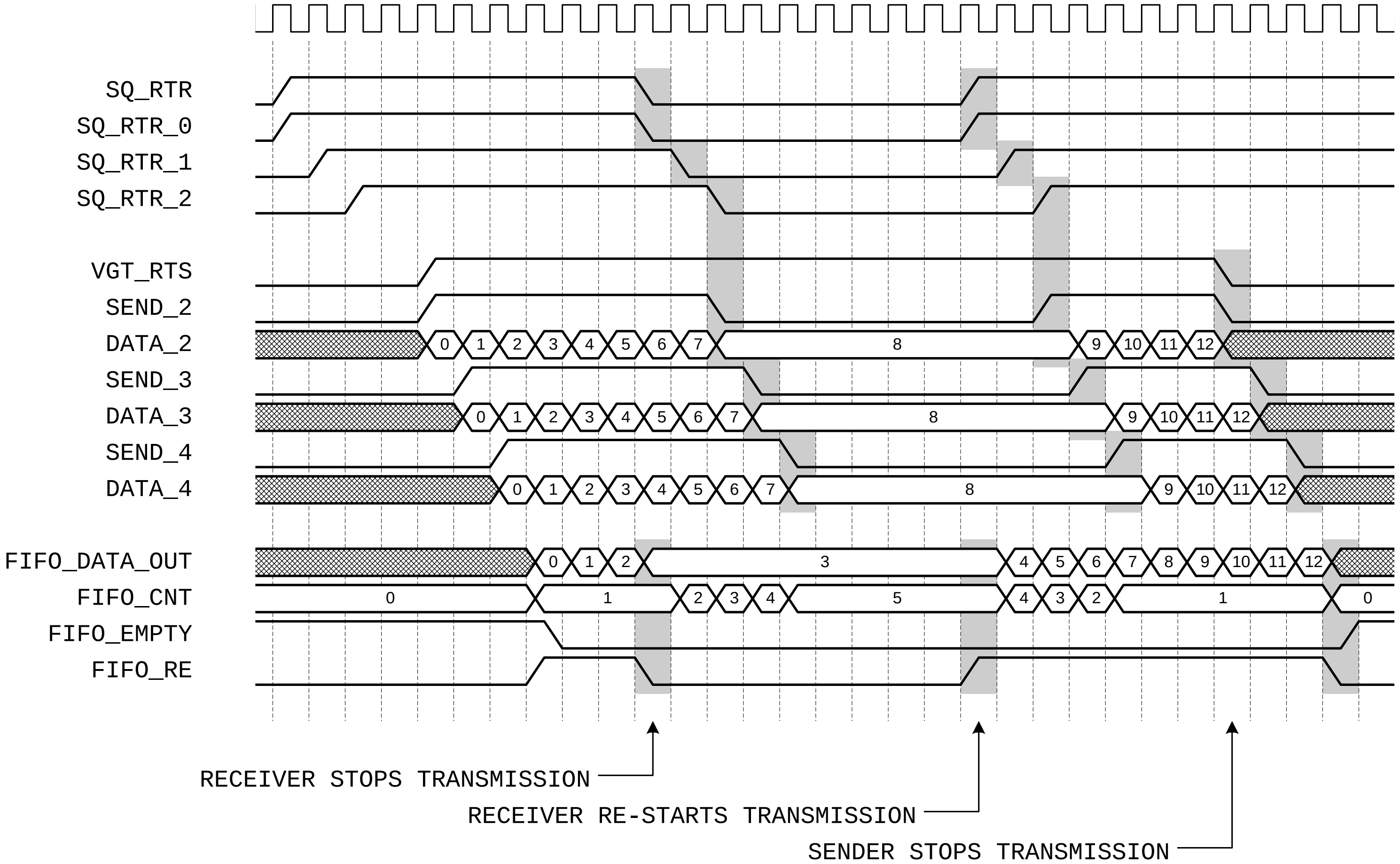


Figure 1. Detailed Logical Diagram for PA_SQ_vgt Interface.

23.2.6 *SQ to SX: Control bus*

Name	**Direction**	**Bits**	**Description**
SQ_SXx_exp_type	SQ→SXx	2	00: Pixel without z (1 to 4 buffers) 01: Pixel with z (1 to 4 buffers) 10: Position (1 or 2 results) 11: Pass thru (4,8 or 12 results aligned)
SQ_SXx_exp_number	SQ→SXx	2	Number of locations needed in the export buffer (encoding depends on the type see bellow).
SQ_SXx_exp_alu_id	SQ→SXx	1	ALU ID
SQ_SXx_exp_valid	SQ→SXx	1	Valid bit
SQ_SXx_exp_state	SQ→SXx	3	State Context
SQ_SXx_free_done	SQ→SXx	1	~~Pulse to indicate that the previous export is finished (this can be sent with or without the other fields of the interface)~~Pulse that indicates that the previous export is finished **from the point of view of the SP. This does not necessarily mean that the data has been transferred to RB or PA, or that the space in export buffer for that particular vector thread has been freed up.**
SQ_SXx_free_alu_id	SQ→SXx	1	ALU ID

Depending on the type the number of export location changes:

- Type 00 : Pixels without Z
    - 00 = 1 buffer
    - 01 = 2 buffers
    - 10 = 3 buffers
    - 11 = 4 buffer
- Type 01: Pixels with Z
    - 00 = 2 Buffers (color + Z)
    - 01 = 3 buffers (2 color + Z)
    - 10 = 4 buffers (3 color + Z)
    - 11 = 5 buffers (4 color + Z)
- Type 10 : Position export
    - 00 = 1 position
    - 01 = 2 positions
    - 1X = Undefined
- Type 11: Pass Thru
    - 00 = 4 buffers
    - 01 = 8 buffers
    - 10 = 12 buffers
    - 11 = Undefined

Below the thick black line is the end of transfer packet that tells the SX that a given export is finished. The report packet **will always arrive either before or at the same time than the next export to the same ALU id**.

23.2.7 *SX to SQ : Output file control*

Name	**Direction**	**Bits**	**Description**
SXx_SQ_exp_count_rdy	SXx→SQ	1	Raised by SX0 to indicate that the following two fields reflect the result of the most recent export
SXx_SQ_exp_pos_avail	SXx→SQ	1	Specifies whether there is room for another position.
SXx_SQ_exp_buf_avail	SXx→SQ	7	Specifies the space available in the output buffers. 0: buffers are full 1: 2K-bits available (32-bits for each of the 64 pixels in a clause)

			... 64: 128K-bits available (16 128-bit entries for each of 64 pixels) 65-127: RESERVED

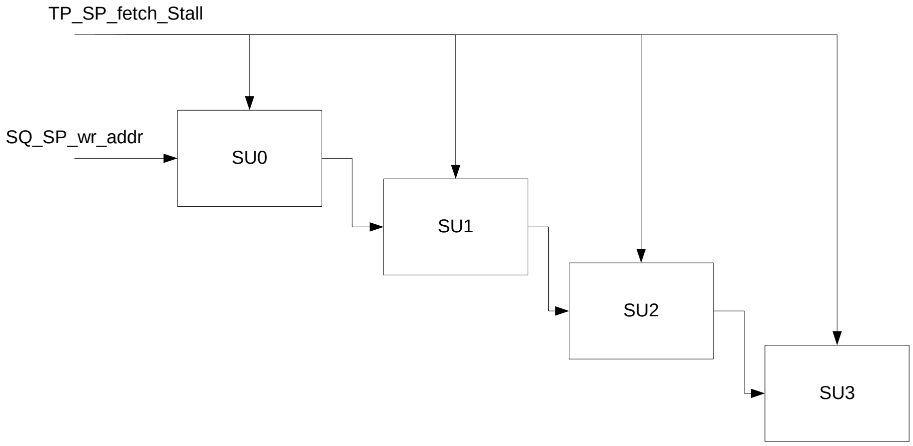
23.2.8 *SQ to TP: Control bus*

Once every clock, the fetch unit sends to the sequencer on which RS line it is now working and if the data in the GPRs is ready or not. This way the sequencer can update the fetch valid bits flags for the reservation station. The sequencer also provides the instruction and constants for the fetch to execute and the address in the register file where to write the fetch return data.

Name	**Direction**	**Bits**	**Description**
TPx_SQ_data_rdy	TPx→ SQ	1	Data ready
TPx_SQ_rs_line_num	TPx→ SQ	6	Line number in the Reservation station
TPx_SQ_type	TPx→ SQ	1	Type of data sent (0:PIXEL, 1:VERTEX)
SQ_TPx_send	SQ→TPx	1	Sending valid data
SQ_TPx_const	SQ→TPx	48	Fetch state sent over 4 clocks (192 bits total)
SQ_TPx_instr	SQ→TPx	24	Fetch instruction sent over 4 clocks
SQ_TPx_end_of_group	SQ→TPx	1	Last instruction of the group
SQ_TPx_Type	SQ→TPx	1	Type of data sent (0:PIXEL, 1:VERTEX)
SQ_TPx_gpr_phase	SQ→TPx	2	Write phase signal
SQ_TP0_lod_correct	SQ→TP0	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP0_pix_mask	SQ→TP0	4	Pixel mask 1 bit per pixel
SQ_TP1_lod_correct	SQ→TP1	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP1_pix_mask	SQ→TP1	4	Pixel mask 1 bit per pixel
SQ_TP2_lod_correct	SQ→TP2	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP2_pix_mask	SQ→TP2	4	Pixel mask 1 bit per pixel
SQ_TP3_lod_correct	SQ→TP3	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP3_pix_mask	SQ→TP3	4	Pixel mask 1 bit per pixel
SQ_TPx_rs_line_num	SQ→TPx	6	Line number in the Reservation station
SQ_TPx_write_gpr_index	SQ->TPx	7	Index into Register file for write of returned Fetch Data

23.2.9 *TP to SQ: Texture stall*

The TP sends this signal to the SQ and the SPs when its input buffer is full.



Name	**Direction**	**Bits**	**Description**

TP_SQ_fetch_stall	TP→ SQ	1	Do not send more texture request if asserted

23.2.10 SQ to SP: Texture stall

Name	Direction	Bits	Description
SQ_SPx_fetch_stall	SQ→SPx	1	Do not send more texture request if asserted

23.2.11 SQ to SP: GPR and auto counter

Name	Direction	Bits	Description
SQ_SPx_gpr_wr_addr	SQ→SPx	7	Write address
SQ_SPx_gpr_rd_addr	SQ→SPx	7	Read address
SQ_SPx_gpr_rd_en	SQ→SPx	1	Read Enable
SQ_SP0_gpr_wr_en	SQ→SPx	1	Write Enable for the GPRs of SP0
SQ_SP1_gpr_wr_en	SQ→SPx	1	Write Enable for the GPRs of SP1
SQ_SP2_gpr_wr_en	SQ→SPx	1	Write Enable for the GPRs of SP2
SQ_SP3_gpr_wr_en	SQ→SPx	1	Write Enable for the GPRs of SP3
SQ_SPx_gpr_phase	SQ→SPx	2	The phase mux (arbitrates between inputs, ALU SRC reads and writes)
SQ_SPx_channel_mask	SQ→SPx	4	The channel mask
SQ_SPx_gpr_input_sel	SQ→SPx	2	When the phase mux selects the inputs this tells from which source to read from: Interpolated data, VTX0, VTX1, autogen counter.
SQ_SPx_auto_count	SQ→SPx	12?	Auto count generated by the SQ, common for all shader pipes

23.2.12 *SQ to SPx: Instructions*

Name	Direction	Bits	Description
SQ_SPx_instr_start	SQ→SPx	1	Instruction start
SQ_SP_instr	SQ→SPx	22	Transferred over 4 cycles 0: SRC A Select 2:0 SRC A Argument Modifier 3:3 SRC A swizzle 11:4 VectorDst 17:12 Per channel use mask (PV/Reg) 21:18 --- - 1: SRC B Select 2:0 SRC B Argument Modifier 3:3 SRC B swizzle 11:4 ScalarDst 17:12 Per channel use mask (PV/Reg) 21:18 --- - 2: SRC C Select 2:0 SRC C Argument Modifier 3:3 SRC C swizzle 11:4 Per channel use mask (PV/Reg) 21:18 --- - 3: Vector Opcode 4:0 Scalar Opcode 10:5 Vector Clamp 11:11 Scalar Clamp 12:12 Vector Write Mask 16:13 Scalar Write Mask 20:17
SQ_SPx_exp_alu_id	SQ→SPx	1	ALU ID
SQ_SPx_exporting	SQ→SPx	1~~2~~	0: Not Exporting 1: ~~Vector~~ Exporting ~~2: Scalar Exporting~~
SQ_SPx_stall	SQ→SPx	1	Stall signal
SQ_SP0_write_mask	SQ→SP0	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP1_ write_mask	SQ→SP1	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP2_ write_mask	SQ→SP2	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP3_ write_mask	SQ→SP3	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SPx_last	SQ→SPx	1	Last instruction of the block
SQ_SP0_pred_overwrite	SQ→SP0	4	Indicates to overwrite the use of PV/PS because of the predication (use the GPRs instead). This operation is done on a per-pixel basis.
SQ_SP1_pred_overwrite	SQ→SP1	4	Indicates to overwrite the use of PV/PS because of the predication (use the GPRs instead). This operation is done on a per-pixel basis.
SQ_SP2_pred_overwrite	SQ→SP2	4	Indicates to overwrite the use of PV/PS because of

			the predication (use the GPRs instead). This operation is done on a per-pixel basis.
SQ_SP3_pred_overwrite	SQ→SP3	4	Indicates to overwrite the use of PV/PS because of the predication (use the GPRs instead). This operation is done on a per-pixel basis.

23.2.13 *SP to SQ: Constant address load/ Predicate Set*

Name	**Direction**	**Bits**	**Description**
SP0_SQ_const_addr	SP0→SQ	36	Constant address load / predicate vector load (4 bits only) to the sequencer
SP0_SQ_valid	SP0→SQ	1	Data valid
SP1_SQ_const_addr	SP1→SQ	36	Constant address load / predicate vector load (4 bits only) to the sequencer
SP1_SQ_valid	SP1→SQ	1	Data valid
SP2_SQ_const_addr	SP2→SQ	36	Constant address load / predicate vector load (4 bits only) to the sequencer
SP2_SQ_valid	SP2→SQ	1	Data valid
SP3_SQ_const_addr	SP3→SQ	36	Constant address load / predicate vector load (4 bits only) to the sequencer
SP3_SQ_valid	SP3→SQ	1	Data valid
SP0_SQ_data_type	SP→SQ	1	Data Type 0: Constant Load 1: Predicate Set

23.2.14 *SQ to SPx: constant broadcast*

Name	**Direction**	**Bits**	**Description**
SQ_SPx_const	SQ→SPx	128	Constant broadcast

23.2.15 *SP0 to SQ: Kill vector load*

Name	**Direction**	**Bits**	**Description**
SP0_SQ_kill_vect	SP0→SQ	4	Kill vector load
SP1_SQ_kill_vect	SP1→SQ	4	Kill vector load
SP2_SQ_kill_vect	SP2→SQ	4	Kill vector load
SP3_SQ_kill_vect	SP3→SQ	4	Kill vector load

23.2.16 *SQ to CP: RBBM bus*

Name	**Direction**	**Bits**	**Description**
SQ_RBB_rs	SQ→CP	1	Read Strobe
SQ_RBB_rd	SQ→CP	32	Read Data
SQ_RBBM_nrtrtr	SQ→CP	1	Optional
SQ_RBBM_rtr	SQ→CP	1	Real-Time (Optional)

23.2.17 *CP to SQ: RBBM bus*

Name	**Direction**	**Bits**	**Description**
rbbm_we	CP→SQ	1	Write Enable
rbbm_a	CP→SQ	15	Address -- Upper Extent is TBD (16:2)
rbbm_wd	CP→SQ	32	Data
rbbm_be	CP→SQ	4	Byte Enables
rbbm_re	CP→SQ	1	Read Enable
rbb_rs0	CP→SQ	1	Read Return Strobe 0
rbb_rs1	CP→SQ	1	Read Return Strobe 1
rbb_rd0	CP→SQ	32	Read Data 0
rbb_rd1	CP→SQ	32	Read Data 0
RBBM_SQ_soft_reset	CP→SQ	1	Soft Reset

23.2.18 *SQ to CP: State report*

Name	Direction	Bits	Description
SQ_CP_vs_event	SQ→CP	1	Vertex Shader Event
SQ_CP_vs_eventid	SQ→CP	2	Vertex Shader Event ID
SQ_CP_ps_event	SQ→CP	1	Pixel Shader Event
SQ_CP_ps_eventid	SQ→CP	2	Pixel Shader Event ID

eventid = 0 => *sEndOfState (i.e. VsEndOfState)
eventid = 1 => *sDone (i.e. VsDone)

So, the CP will assume the Vs is done with a state whenever it gets a pulse on the SQ_CP_vs_event and the SQ_CP_vs_eventid = 0.

23.3 Example of control flow program execution

We now provide some examples of execution to better illustrate the new design.

Given the program:

Alu 0
Alu 1
Tex 0
Tex 1
Alu 3 Serial
Alu 4
Tex 2
Alu 5
Alu 6 Serial
Tex 3
Alu 7
Alloc Position 1 buffer
Alu 8 Export
Tex 4
Alloc Parameter 3 buffers
Alu 9 Export 0
Tex 5
Alu 10 Serial Export 2
Alu 11 Export 1 End

Would be converted into the following CF instructions:

```
Execute 0 Alu 0 Alu 0 Tex 0 Tex 1 Alu 0 Alu 0 Tex 0 Alu 1 Alu 0 Tex
Execute 0 Alu
Alloc Position 1
Execute 0 Alu 0 Tex
Alloc Param 3
Execute_end 0 Alu 0 Tex 1 Alu 0 Alu
```

And the execution of this program would look like this:

Put thread in Vertex RS:

Control Flow Instruction Pointer (12 bits), (CFP)
Execution Count Marker (3 or 4 bits), (ECM)
Loop Iterators (4x9 bits), (LI)
Call return pointers (4x12 bits), (CRP)
Predicate Bits(4x64 bits), (PB)
Export ID (1 bit), (EXID)

GPR Base Ptr (8 bits), (GPR)
Export Base Ptr (7 bits), (EB)
Context Ptr (3 bits).(CPTR)
LOD correction bits (16x6 bits) (LOD)

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	0	0	0	0	0	0	0	0	0

Valid Thread (VALID)
Texture/ALU engine needed (TYPE)
Texture Reads are outstanding (PENDING)
Waiting on Texture Read to Complete (SERIAL)
Allocation Wait (2 bits) (ALLOC)
    00 – No allocation needed
    01 – Position export allocation needed (ordered export)
    10 – Parameter or pixel export needed (ordered export)
    11 – pass thru (out of order export)
Allocation Size (4 bits) (SIZE)
Position Allocated (POS_ALLOC)
First thread of a new context (FIRST)
Last (1 bit), (LAST)

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	0	0	0	0	0	1	0

Then the thread is picked up for the execution of the first control flow instruction:

```
Execute 0 Alu 0 Alu 0 Tex 0 Tex 1 Alu 0 Alu 0 Tex 0 Alu 1 Alu 0 Tex
```

It executes the first two ALU instructions and goes back to the RS for a resource request change. Here is the state returned to the RS:

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	2	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	0	0	0	0	0	1	0

Then when the texture pipe frees up, the arbiter picks up the thread to issue the texture reads. The thread comes back in this state:

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	4	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	1	0	0	0	1	0

Because of the serial bit the arbiter must wait for the texture to return and clear the PENDING bit before it can pick the thread up. Lets say that the texture reads are complete, then the arbiter picks up the thread and returns it in this state:

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	6	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	0	0	0	0	0	1	0

Again the TP frees up, the arbiter picks up the thread and executes. It returns in this state:

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	7	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	0	0	0	1	0

Now, even if the texture has not returned we can still pick up the thread for ALU execution because the serial bit is not set. The thread will however come back to the RS for the second ALU instruction because it has the serial bit set.

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	8	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	1	0	0	0	1	0

As soon as the TP clears the pending bit the thread is picked up and returns:

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	9	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	0	0	0	0	0	1	0

Picked up by the TP and returns:

```
Execute 0 Alu
```

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
1	0	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	0	0	0	1	0

Picked up by the ALU and returns (lets say the TP has not returned yet):

```
Alloc Position 1
```

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
2	0	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	01	1	0	1	0

If the SX has the place for the export, the SQ is going to allocate and pick up the thread for execution. It returns to the RS in this state:

```
Execute 0 Alu 0 Tex
```

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
3	1	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	1	0	0	0	1	1	0

Now, since the TP has not returned yet, we must wait for it to return because we cannot issue multiple texture requests. The TP returns, clears the PENDING bit and we proceed:

```
Alloc Param 3
```

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
4	0	0	0	0	1	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	10	3	1	1	0

Once again the SQ makes sure the SX has enough room in the Parameter cache before it can pick up this thread.

```
Execute_end 0 Alu 0 Tex 1 Alu 0 Alu
```

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
5	1	0	0	0	1	0	100	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	1	0	0	0	1	1	0

This executes on the TP and then returns:

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
5	2	0	0	0	1	0	100	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	1	0	0	1	1	1

Waits for the TP to return because of the textures reads are pending (and SERIAL in this case). Then executes and does not return to the RS because the LAST bit is set. This is the end of this thread and before dropping it on the floor, the SQ notifies the SX of export completion.

24. Open issues

Need to do some testing on the size of the register file as well as on the register file allocation method (dynamic VS static).

Saving power?

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 2015~~9~~ September, 20022	GEN-CXXXXX-REVA	1 of 53

Author: Laurent Lefebvre

Issue To:	**Copy No:**

R400 Sequencer Specification

SQ

Version 2.054

Overview: This is an architectural specification for the R400 Sequencer block (SEQ). It provides an overview of the required capabilities and expected uses of the block. It also describes the block interfaces, internal sub-blocks, and provides internal state diagrams.

AUTOMATICALLY UPDATED FIELDS:

Document Location:	C:\perforce\r400\doc_lib\design\blocks\sq\R400_Sequencer.doc
Current Intranet Search Title:	R400 Sequencer Specification

APPROVALS	
Name/Dept	Signature/Date

Remarks:

THIS DOCUMENT CONTAINS CONFIDENTIAL INFORMATION THAT COULD BE SUBSTANTIALLY DETRIMENTAL TO THE INTEREST OF ATI TECHNOLOGIES INC. THROUGH UNAUTHORIZED USE OR DISCLOSURE.

ATI 2033
LG v. ATI
IPR2015-00326

ATI Ex. 2119
IPR2023-00922

Table Of Contents

1. OVERVIEW .... **9~~7~~**
1.1 Top Level Block Diagram .... 11~~9~~
1.2 Data Flow graph (SP) .... 12~~10~~
1.3 Control Graph .... 13~~11~~
2. INTERPOLATED DATA BUS .... **13~~11~~**
3. INSTRUCTION STORE .... **16~~14~~**
4. SEQUENCER INSTRUCTIONS .... **16~~14~~**
5. CONSTANT STORES .... **16~~14~~**
5.1 Memory organizations .... 16~~14~~
5.2 Management of the Control Flow Constants .... 17~~15~~
5.3 Management of the re-mapping tables .... 17~~15~~
5.3.1 R400 Constant management .... 17~~15~~
5.3.2 Proposal for R400LE constant management .... 17~~15~~
5.3.3 Dirty bits .... 19~~17~~
5.3.4 Free List Block .... 19~~17~~
5.3.5 De-allocate Block .... 20~~18~~
5.3.6 Operation of Incremental model .... 20~~18~~
5.4 Constant Store Indexing .... 20~~18~~
5.5 Real Time Commands .... 21~~19~~
5.6 Constant Waterfalling .... 21~~19~~
6. LOOPING AND BRANCHES .... **22~~20~~**
6.1 The controlling state. .... 22~~20~~
6.2 The Control Flow Program .... 22~~20~~
6.2.1 Control flow instructions table .... 23~~21~~
6.3 Implementation .... 25~~23~~
6.4 Data dependant predicate instructions .... 27~~24~~
6.5 HW Detection of PV,PS .... 27~~25~~
6.6 Register file indexing .... 27~~25~~
6.7 Debugging the Shaders .... 28~~25~~
6.7.1 Method 1: Debugging registers .... 28~~26~~
6.7.2 Method 2: Exporting the values in the GPRs .... 28~~26~~
7. PIXEL KILL MASK .... **28~~26~~**
8. MULTIPASS VERTEX SHADERS (HOS) .... **29~~26~~**
9. REGISTER FILE ALLOCATION .... **29~~26~~**
10. FETCH ARBITRATION .... **30~~28~~**
11. ALU ARBITRATION .... **30~~28~~**
12. HANDLING STALLS .... **31~~29~~**
13. CONTENT OF THE RESERVATION STATION FIFOS .... **31~~29~~**
14. THE OUTPUT FILE .... **31~~29~~**
15. IJ FORMAT .... **31~~29~~**
15.1 Interpolation of constant attributes .... 31~~29~~
16. STAGING REGISTERS .... **32~~30~~**

17. THE PARAMETER CACHE .... 33~~31~~
17.1 Export restrictions .... 34~~32~~
17.1.1 Pixel exports: .... 34~~32~~
17.1.2 Vertex exports: .... 34~~32~~
17.1.3 Pass thru exports: .... 34~~32~~
17.2 Arbitration restrictions .... 34~~32~~
18. EXPORT TYPES .... 34~~32~~
18.1 Vertex Shading .... 34~~32~~
18.2 Pixel Shading .... 35~~33~~
19. SPECIAL INTERPOLATION MODES .... 35~~33~~
19.1 Real time commands .... 35~~33~~
19.2 Sprites/ XY screen coordinates/ FB information .... 35~~33~~
19.3 Auto generated counters .... 36~~34~~
19.3.1 Vertex shaders .... 36~~34~~
19.3.2 Pixel shaders .... 36~~34~~
20. STATE MANAGEMENT .... 37~~35~~
20.1 Parameter cache synchronization .... 37~~35~~
21. XY ADDRESS IMPORTS .... 37~~35~~
21.1 Vertex indexes imports .... 37~~35~~
22. REGISTERS .... 37~~35~~
23. INTERFACES .... 38~~36~~
23.1 External Interfaces .... 38~~36~~
23.2 SC to SP Interfaces .... 38~~36~~
23.2.1 SC_SP# .... 38~~36~~
23.2.2 SC_SQ .... 39~~37~~
23.2.3 SQ to SX(SP): Interpolator bus .... 41~~39~~
23.2.4 SQ to SP: Staging Register Data .... 41~~39~~
23.2.5 VGT to SQ : Vertex interface .... 41~~39~~
23.2.6 SQ to SX: Control bus .... 44~~42~~
23.2.7 SX to SQ : Output file control .... 44~~42~~
23.2.8 SQ to TP: Control bus .... 45~~43~~
23.2.9 TP to SQ: Texture stall .... 45~~43~~
23.2.10 SQ to SP: Texture stall .... 46~~44~~
23.2.11 SQ to SP: GPR and auto counter .... 46~~44~~
23.2.12 SQ to SPx: Instructions .... 47~~45~~
23.2.13 SP to SQ: Constant address load/ Predicate Set/Kill set .... 48~~46~~
23.2.14 SQ to SPx: constant broadcast .... 48~~46~~
23.2.15 SQ to CP: RBBM bus .... 49~~46~~
23.2.16 CP to SQ: RBBM bus .... 49~~46~~
23.2.17 SQ to CP: State report .... 49~~47~~
23.3 Example of control flow program execution .... 49~~47~~

24. OPEN ISSUES .... 53~~51~~

~~1. OVERVIEW .... 7~~

~~1.1 Top Level Block Diagram .... 9~~

~~1.2 Data Flow graph (SP) .... 10~~

~~1.3 Control Graph .... 11~~

~~2. INTERPOLATED DATA BUS .... 11~~

~~3. INSTRUCTION STORE .... 14~~

~~4. SEQUENCER INSTRUCTIONS .... 14~~

~~5. CONSTANT STORES .... 14~~

~~5.1 Memory organizations .... 14~~

~~5.2 Management of the Control Flow Constants .... 15~~

~~5.3 Management of the re-mapping tables .... 15~~

~~5.3.1 R400 Constant management .... 15~~

~~5.3.2 Proposal for R400LE constant management .... 15~~

~~5.3.3 Dirty bits .... 17~~

~~5.3.4 Free List Block .... 17~~

~~5.3.5 De-allocate Block .... 18~~

~~5.3.6 Operation of Incremental model .... 18~~

~~5.4 Constant Store Indexing .... 18~~

~~5.5 Real Time Commands .... 19~~

~~5.6 Constant Waterfalling .... 19~~

~~6. LOOPING AND BRANCHES .... 20~~

~~6.1 The controlling state. .... 20~~

~~6.2 The Control Flow Program .... 20~~

~~6.2.1 Control flow instructions table .... 21~~

~~6.3 Implementation .... 23~~

~~6.4 Data dependant predicate instructions .... 24~~

~~6.5 HW Detection of PV,PS .... 25~~

~~6.6 Register file indexing .... 25~~

~~6.7 Debugging the Shaders .... 25~~

~~6.7.1 Method 1: Debugging registers .... 26~~

~~6.7.2 Method 2: Exporting the values in the GPRs .... 26~~

~~7. PIXEL KILL MASK .... 26~~

~~8. MULTIPASS VERTEX SHADERS (HOS) .... 26~~

~~9. REGISTER FILE ALLOCATION .... 26~~

~~10. FETCH ARBITRATION .... 28~~

~~11. ALU ARBITRATION .... 28~~

~~12. HANDLING STALLS .... 29~~

~~13. CONTENT OF THE RESERVATION STATION FIFOS .... 29~~

~~14. THE OUTPUT FILE .... 29~~

~~15. IJ FORMAT .... 29~~

~~15.1 Interpolation of constant attributes .... 29~~

~~16. STAGING REGISTERS .... 30~~

~~17. THE PARAMETER CACHE .... 31~~

~~17.1 Export restrictions .... 32~~

~~17.1.1 Pixel exports: 32~~

~~17.1.2 Vertex exports: 32~~

~~17.1.3 Pass thru exports: 32~~

~~17.2 Arbitration restrictions 32~~

~~**18. EXPORT TYPES 32**~~

~~18.1 Vertex Shading 32~~

~~18.2 Pixel Shading 33~~

~~**19. SPECIAL INTERPOLATION MODES 33**~~

~~19.1 Real time commands 33~~

~~19.2 Sprites/ XY screen coordinates/ FB information 33~~

~~19.3 Auto generated counters 34~~

~~19.3.1 Vertex shaders 34~~

~~19.3.2 Pixel shaders 34~~

~~**20. STATE MANAGEMENT 34**~~

~~20.1 Parameter cache synchronization 34~~

~~**21. XY ADDRESS IMPORTS 35**~~

~~21.1 Vertex indexes imports 35~~

~~**22. REGISTERS 35**~~

~~22.1 Control **Error! Bookmark not defined.**~~

~~22.2 Context **Error! Bookmark not defined.**~~

~~**23. DEBUG REGISTERS** ERROR! BOOKMARK NOT DEFINED.~~

~~23.1 Context **Error! Bookmark not defined.**~~

~~23.2 Control **Error! Bookmark not defined.**~~

~~**24. INTERFACES 35**~~

~~24.1 External Interfaces 35~~

~~24.2 SC to SP Interfaces 35~~

~~24.2.1 SC_SP# 35~~

~~24.2.2 SC_SQ 36~~

~~24.2.3 SQ to SX: Interpolator bus 38~~

~~24.2.4 SQ to SP: Staging Register Data 38~~

~~24.2.5 VGT to SQ : Vertex interface 38~~

~~24.2.6 SQ to SX: Control bus 41~~

~~24.2.7 SX to SQ : Output file control 41~~

~~24.2.8 SQ to TP: Control bus 42~~

~~24.2.9 TP to SQ: Texture stall 42~~

~~24.2.10 SQ to SP: Texture stall 43~~

~~24.2.11 SQ to SP: GPR and auto counter 43~~

~~24.2.12 SQ to SPx: Instructions 44~~

~~24.2.13 SP to SQ: Constant address load/ Predicate Set 45~~

~~24.2.14 SQ to SPx: constant broadcast 45~~

~~24.2.15 SP0 to SQ: Kill vector load 45~~

~~24.2.16 SQ to CP: RBBM bus .... 45~~

~~24.2.17 CP to SQ: RBBM bus .... 45~~

~~24.2.18 SQ to CP: State report .... 45~~

~~24.3 Example of control flow program execution .... 46~~

~~**25. OPEN ISSUES .... 50**~~

Revision Changes:

Rev 0.1 (Laurent Lefebvre) Date: May 7, 2001	First draft.
Rev 0.2 (Laurent Lefebvre) Date : July 9, 2001	Changed the interfaces to reflect the changes in the SP. Added some details in the arbitration section.
Rev 0.3 (Laurent Lefebvre) Date : August 6, 2001	Reviewed the Sequencer spec after the meeting on August 3, 2001.
Rev 0.4 (Laurent Lefebvre) Date : August 24, 2001	Added the dynamic allocation method for register file and an example (written in part by Vic) of the flow of pixels/vertices in the sequencer.
Rev 0.5 (Laurent Lefebvre) Date : September 7, 2001	Added timing diagrams (Vic)
Rev 0.6 (Laurent Lefebvre) Date : September 24, 2001	Changed the spec to reflect the new R400 architecture. Added interfaces.
Rev 0.7 (Laurent Lefebvre) Date : October 5, 2001	Added constant store management, instruction store management, control flow management and data dependant predication.
Rev 0.8 (Laurent Lefebvre) Date : October 8, 2001	Changed the control flow method to be more flexible. Also updated the external interfaces.
Rev 0.9 (Laurent Lefebvre) Date : October 17, 2001	Incorporated changes made in the 10/18/01 control flow meeting. Added a NOP instruction, removed the conditional_execute_or_jump. Added debug registers.
Rev 1.0 (Laurent Lefebvre) Date : October 19, 2001	Refined interfaces to RB. Added state registers.
Rev 1.1 (Laurent Lefebvre) Date : October 26, 2001	Added SEQ→SP0 interfaces. Changed delta precision. Changed VGT→SP0 interface. Debug Methods added.
Rev 1.2 (Laurent Lefebvre) Date : November 16, 2001	Interfaces greatly refined. Cleaned up the spec.
Rev 1.3 (Laurent Lefebvre) Date : November 26, 2001	Added the different interpolation modes.
Rev 1.4 (Laurent Lefebvre) Date : December 6, 2001	Added the auto incrementing counters. Changed the VGT→SQ interface. Added content on constant management. Updated GPRs.
Rev 1.5 (Laurent Lefebvre) Date : December 11, 2001	Removed from the spec all interfaces that weren't directly tied to the SQ. Added explanations on constant management. Added PA→SQ synchronization fields and explanation.
Rev 1.6 (Laurent Lefebvre) Date : January 7, 2002	Added more details on the staging register. Added detail about the parameter caches. Changed the call instruction to a Conditionnal_call instruction. Added details on constant management and updated the diagram.
Rev 1.7 (Laurent Lefebvre) Date : February 4, 2002	Added Real Time parameter control in the SX interface. Updated the control flow section.
Rev 1.8 (Laurent Lefebvre) Date : March 4, 2002	New interfaces to the SX block. Added the end of clause modifier, removed the end of clause instructions.
Rev 1.9 (Laurent Lefebvre) Date : March 18, 2002	Rearangement of the CF instruction bits in order to ensure byte alignement.
Rev 1.10 (Laurent Lefebvre) Date : March 25, 2002	Updated the interfaces and added a section on exporting rules.
Rev 1.11 (Laurent Lefebvre) Date : April 19, 2002	Added CP state report interface. Last version of the spec with the old control flow scheme
Rev 2.0 (Laurent Lefebvre) Date : April 19, 2002	New control flow scheme

Rev 2.01 (Laurent Lefebvre) Date : May 2, 2002	Changed slightly the control flow instructions to allow force jumps and calls.
Rev 2.02 (Laurent Lefebvre) Date : May 13, 2002	Updated the Opcodes. Added type field to the constant/pred interface. Added Last field to the SQ→SP instruction load interface.
Rev 2.03 (Laurent Lefebvre) Date : July 15, 2002	SP interface updated to include predication optimizations. Added the predicate no stall instructions,
Rev 2.04 (Laurent Lefebvre) Date :August 2, 2002	Documented the new parameter generation scheme for XY coordinates points and lines STs.
Rev 2.05 (Laurent Lefebvre) Date :	

1. Overview

The sequencer chooses two ALU threads and a fetch hread to execute, and executes all of the instructions in a block before looking for a new clause of the same type. Two ALU threads are executed interleaved to hide the ALU latency. The arbitrator will give priority to older threads. There are two separate reservation stations, one for pixel vectors and one for vertices vectors. This way a pixel can pass a vertex and a vertex can pass a pixel.

To support the shader pipe the sequencer also contains the shader instruction cache, constant store, control flow constants and texture state. The four shader pipes also execute the same instruction thus there is only one sequencer for the whole chip.

The sequencer first arbitrates between vectors of 64 vertices that arrive directly from primitive assembly and vectors of 16 quads (64 pixels) that are generated in the scan converter.

The vertex or pixel program specifies how many GPRs it needs to execute. The sequencer will not start the next vector until the needed space is available in the GPRs.

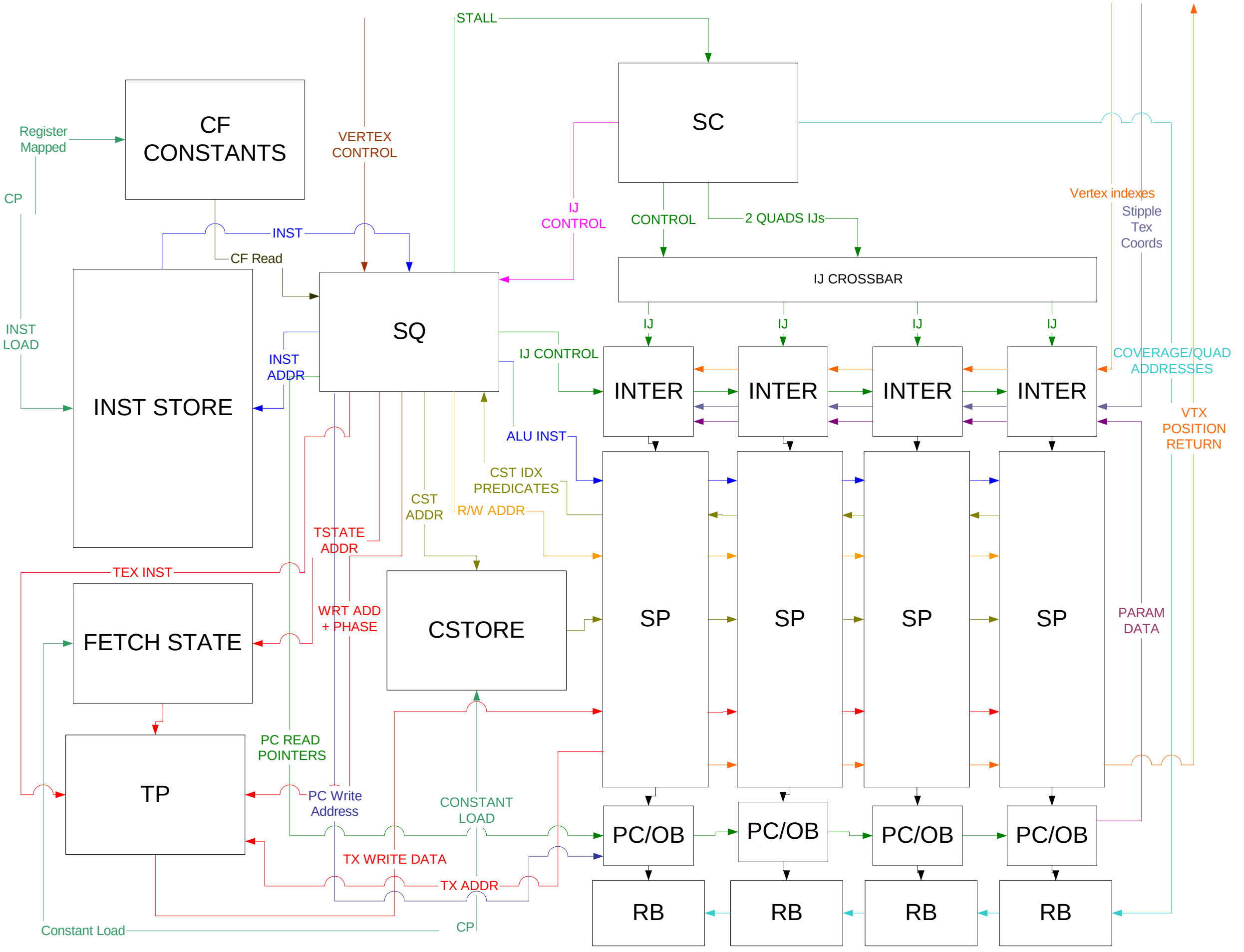


Figure 1: General Sequencer overview

1.1 Top Level Block Diagram

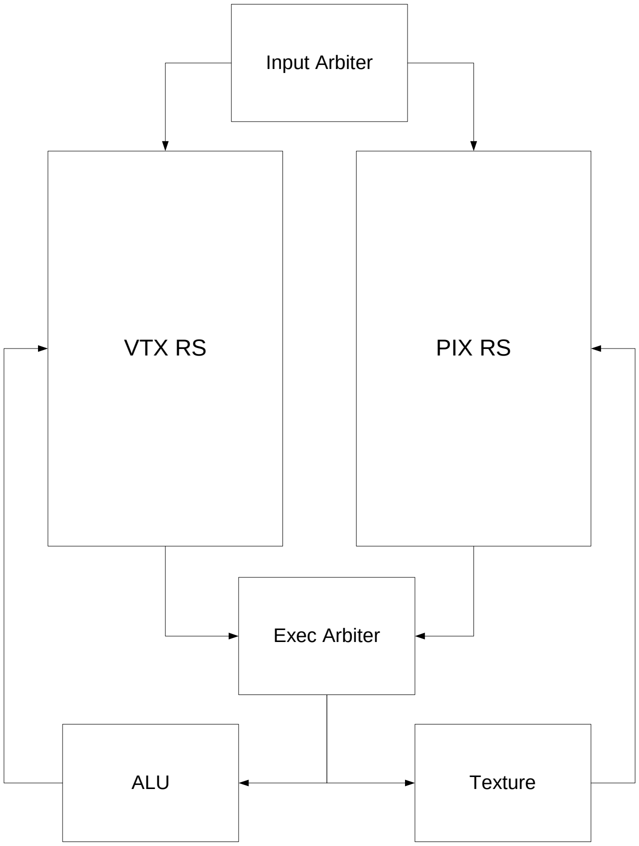


Figure 2: Reservation stations and arbiters

Under this new scheme, the sequencer (SQ) will only use one global state management machine per vector type (pixel, vertex) that we call the reservation station (RS).

1.2 Data Flow graph (SP)

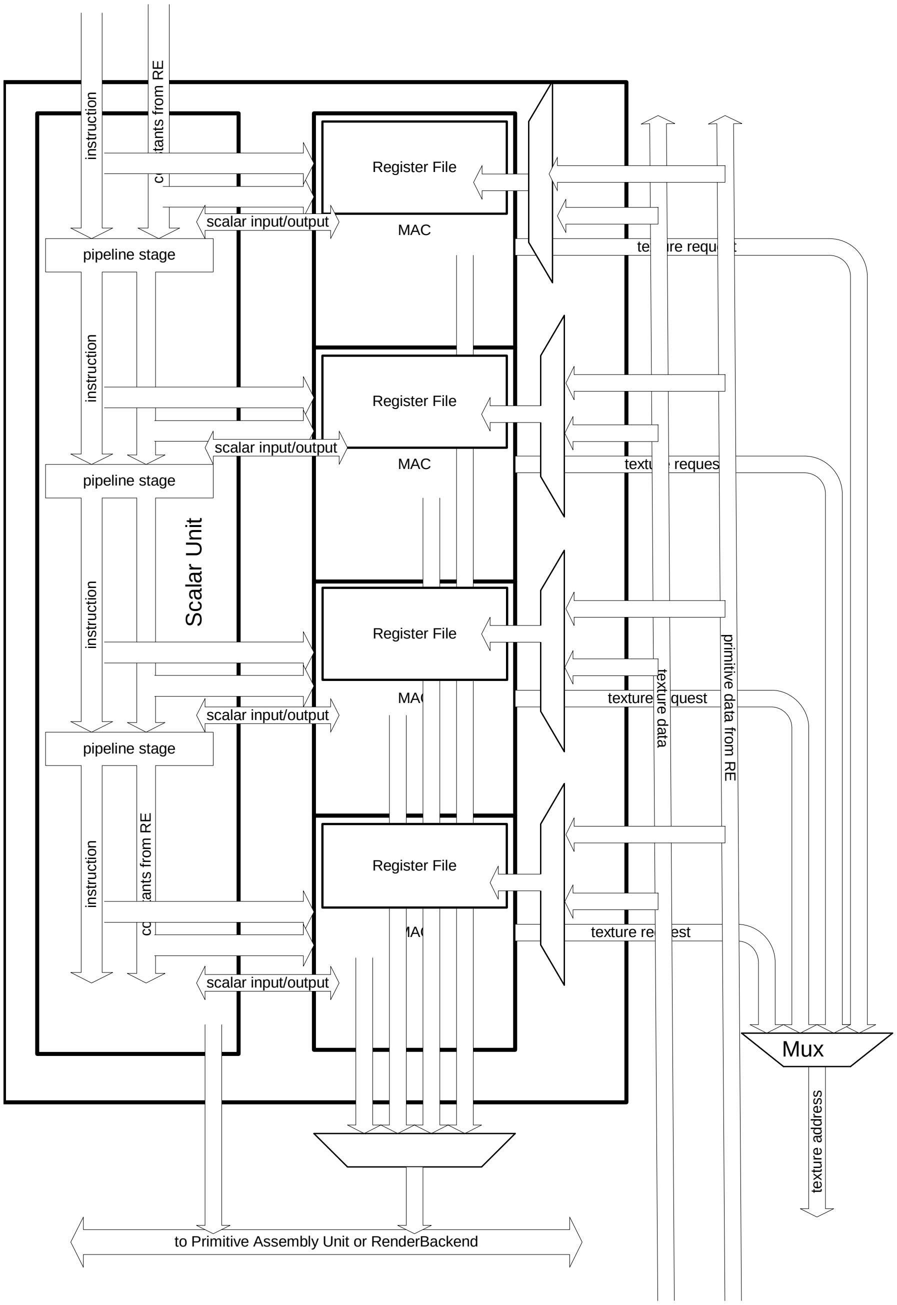


Figure 3: The shader Pipe

The gray area represents blocks that are replicated 4 times per shader pipe (16 times on the overall chip).

1.3 Control Graph

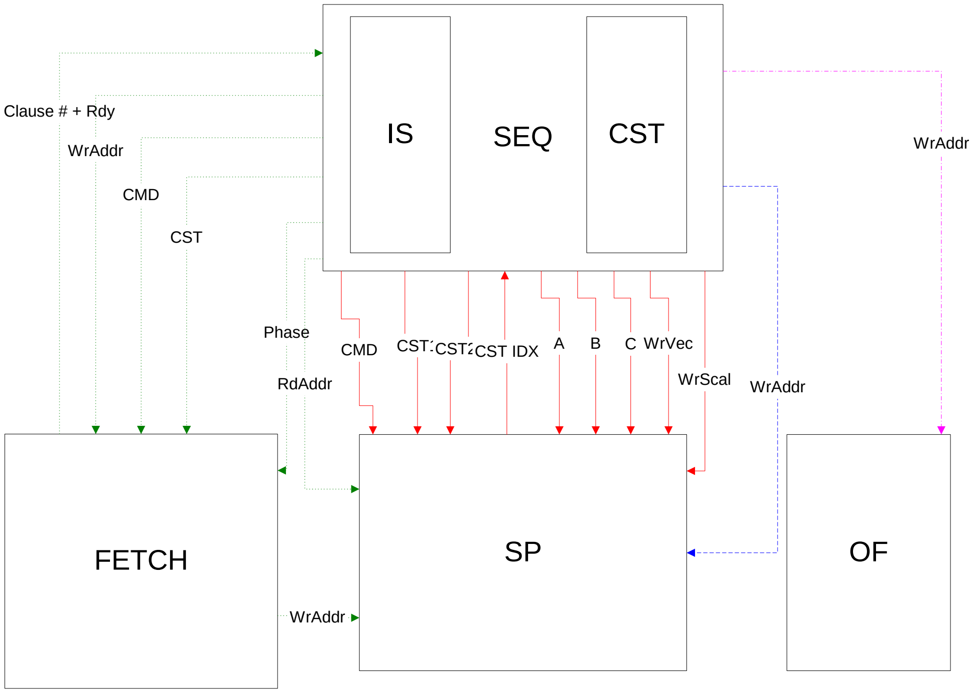


Figure 4: Sequencer Control interfaces

In green is represented the Fetch control interface, in red the ALU control interface, in blue the Interpolated/Vector control interface and in purple is the output file control interface.

2. Interpolated data bus

The interpolators contain an IJ buffer to pack the information as much as possible before writing it to the register file.

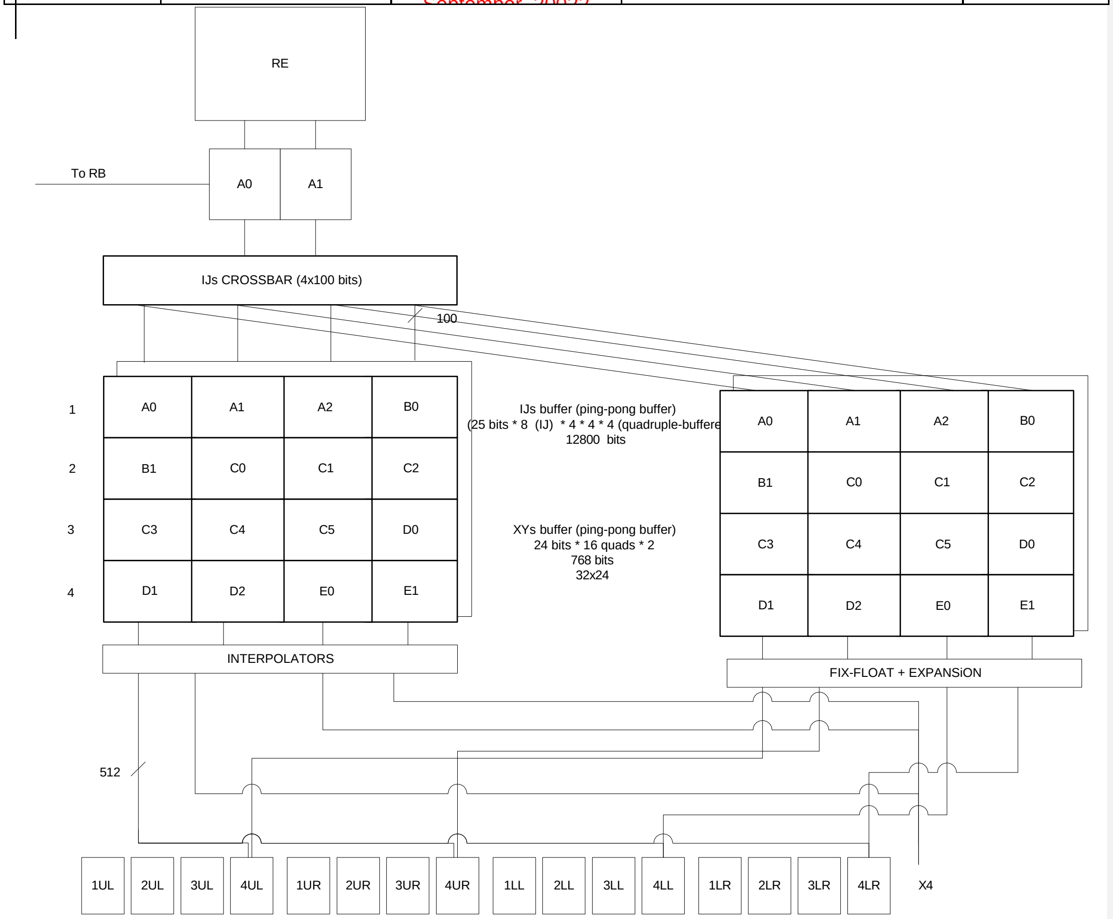


Figure 5: Interpolation buffers

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21	T22	T23
												WRITES												
SP 0	A0	A0	XY A0	B1	B1	XY B1	C3	C3	XY C3				D1	D1	XY D1									
SP 1	A1	A1	XY A1				C0	C0	XY C0	C4	C4	XY C4	D2	D2	XY D2									
SP 2	A2	A2	XY A2				C1	C1	XY C1	C5	C5	XY C5				E0	E0	XY E0						
SP 3				B0	B0	XY B0	C2	C2	XY C2				D0	D0	XY D0	E1	E1	XY E1						
												READS												
SP 0	XY 0-3	XY 16-19	XY 32-35	XY 48-51	A0	B1	C3	D1					A0	B1	C3	D1					V 0-3	V 16-19	V 32-35	V 48-51
SP 1	XY 4-7	XY 20-23	XY 36-39	XY 52-55	A1		C4	D2		C0			A1		C4	D2		C0			V 4-7	V 20-23	V 36-39	V 52-55
SP 2	XY 8-11	XY 24-27	XY 40-43	XY 56-59	A2		C5			C1		E0	A2		C5			C1		E0	V 8-11	V 24-27	V 40-43	V 56-59
SP 3	XY 12-15	XY 28-31	XY 44-47	XY 60-63					B0	C2	D0	E1					B0	C2	D0	E1	V 12-15	V 28-31	V 44-47	V 60-63
			XY					P1									P2						VTX	

Figure 6: Interpolation timing diagram

Above is an example of a tile the sequencer might receive from the SC. The write side is how the data get stacked into the XY and IJ buffers, the read side is how the data is passed to the GPRs. The IJ information is packed in the IJ buffer 4 quads at a time or two clocks. The sequencer allows at any given time as many as four quads to interpolate a parameter. They all have to come from the same primitive. Then the sequencer controls the write mask to the GPRs to write the valid data in.

3. Instruction Store

There is going to be only one instruction store for the whole chip. It will contain 4096 instructions of 96 bits each.

It is likely to be a 1 port memory; we use 1 clock to load the ALU instruction, 1 clocks to load the Fetch instruction, 1 clock to load 2 control flow instructions and 1 clock to write instructions.

The instruction store is loaded by the CP thru the register mapped registers.

The VS_BASE and PS_BASE context registers are used to specify for each context where its shader is in the instruction memory.

For the Real time commands the story is quite the same but for some small differences. There are no wrap-around points for real time so the driver must be careful not to overwrite regular shader data. The shared code (shared subroutines) uses the same path as real time.

4. Sequencer Instructions

All control flow instructions and move instructions are handled by the sequencer only. The ALUs will perform NOPs during this time (MOV PV,PV, PS,PS) if they have nothing else to do.

5. Constant Stores

5.1 Memory organizations

A likely size for the ALU constant store is 1024x128 bits. The read BW from the ALU constant store is 128 bits/clock and the write bandwidth is 32 bits/clock (directed by the CP bus size not by memory ports).

The maximum logical size of the constant store for a given shader is 256 constants. Or 512 for the pixel/vertex shader pair. The size of the re-mapping table is 128 lines (each line addresses 4 constants). The write granularity is 4 constants or 512 bits. It takes 16 clocks to write the four constants. Real time requires 256 lines in the physical memory (this is physically register mapped).

The texture state is also kept in a similar memory. The size of this memory is 320x96 bits (128 texture states for regular mode, 32 states for RT). The memory thus holds 128 texture states (192 bits per state). The logical size exposes 32 different states total, which are going to be shared between the pixel and the vertex shader. The size of the re-mapping table to for the texture state memory is 32 lines (each line addresses 1 texture state lines in the real memory). The CP write granularity is 1 texture state lines (or 192 bits). The driver sends 512 bits but the CP ignores the top 320 bits. It thus takes 6 clocks to write the texture state. Real time requires 32 lines in the physical memory (this is physically register mapped).

The control flow constant memory doesn't sit behind a renaming table. It is register mapped and thus the driver must reload its content each time there is a change in the control flow constants. Its size is 320*32 because it must hold 8 copies of the 32 dwords of control flow constants and the loop construct constants must be aligned.

The constant re-mapping tables for texture state and ALU constants are logically register mapped for regular mode and physically register mapped for RT operation.

5.2 Management of the Control Flow Constants

The control flow constants are register mapped, thus the CP writes to the according register to set the constant, the SQ decodes the address and writes to the block pointed by its current base pointer (CF_WR_BASE). On the read side, one level of indirection is used. A register (SQ_CONTEXT_MISC.CF_RD_BASE) keeps the current base pointer to the control flow block. This register is copied whenever there is a state change. Should the CP write to CF after the state change, the base register is updated with the (current pointer number +1)% number of states. This way, if the CP doesn't write to CF the state is going to use the previous CF constants.

5.3 Management of the re-mapping tables

5.3.1 *R400 Constant management*

The sequencer is responsible to manage two re-mapping tables (one for the constant store and one for the texture state). On a state change (by the driver), the sequencer will broadside copy the contents of its re-mapping tables to a new one. We have 8 different re-mapping tables we can use concurrently.

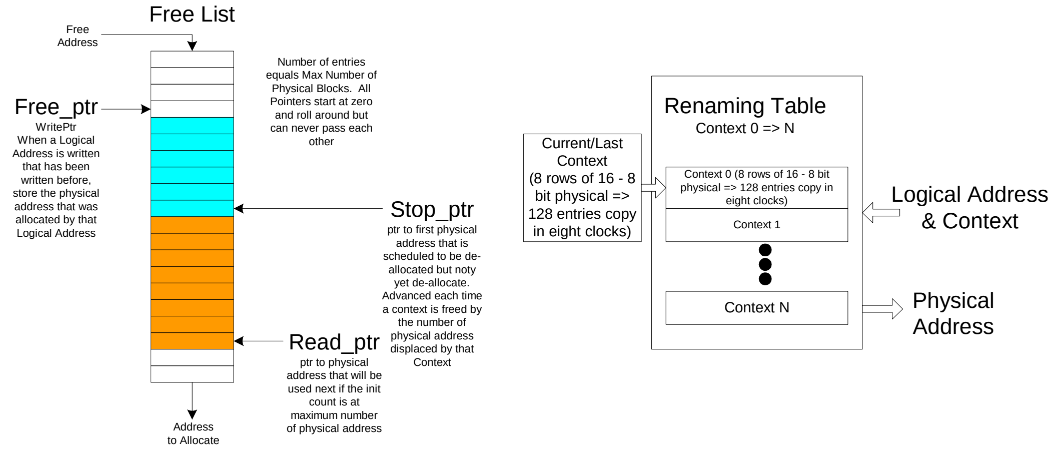
The constant memory update will be incremental, the driver only need to update the constants that actually changed between the two state changes.

For this model to work in its simplest form, the requirement is that the physical memory MUST be at least twice as large as the logical address space + the space allocated for Real Time. In our case, since the logical address space is 512 and the reserved RT space can be up to 256 entries, the memory must be of sizes 1280 and above. Similarly the size of the texture store must be of 32*2+32 = 96 entries and above.

5.3.2 *Proposal for R400LE constant management*

To make this scheme work with only 512+256 = 768 entries, upon reception of a CONTROL packet of state + 1, the sequencer would check for SQ_IDLE and PA_IDLE and if both are idle will erase the content of state to replace it with the new state (this is depicted in Figure 8: De-allocation mechanism~~Figure 8: De-allocation mechanismFigure 8: De-allocation mechanism~~). Note that in the case a state is cleared a value of 0 is written to the corresponding de-allocation counter location so that when the SQ is going to report a state change, nothing will be de-allocated upon the first report.

The second path sets all context dirty bits that were used in the current state to 1 (thus allowing the new state to reuse these physical addresses if needed).


Free List
Free Address
Free_ptr
WritePtr When a Logical Address is written that has been written before, store the physical address that was allocated by that Logical Address
Number of entries equals Max Number of Physical Blocks. All Pointers start at zero and roll around but can never pass each other
Stop_ptr
ptr to first physical address that is scheduled to be de-allocated but noty yet de-allocate. Advanced each time a context is freed by the number of physical address displaced by that Context
Read_ptr
ptr to physical address that will be used next if the init count is at maximum number of physical address
Address to Allocate
Current/Last Context (8 rows of 16 - 8 bit physical => 128 entries copy in eight clocks)
Renaming Table
Context 0 => N
Context 0 (8 rows of 16 - 8 bit physical => 128 entries copy in eight clocks)
Context 1
Context N
Logical Address & Context
Physical Address

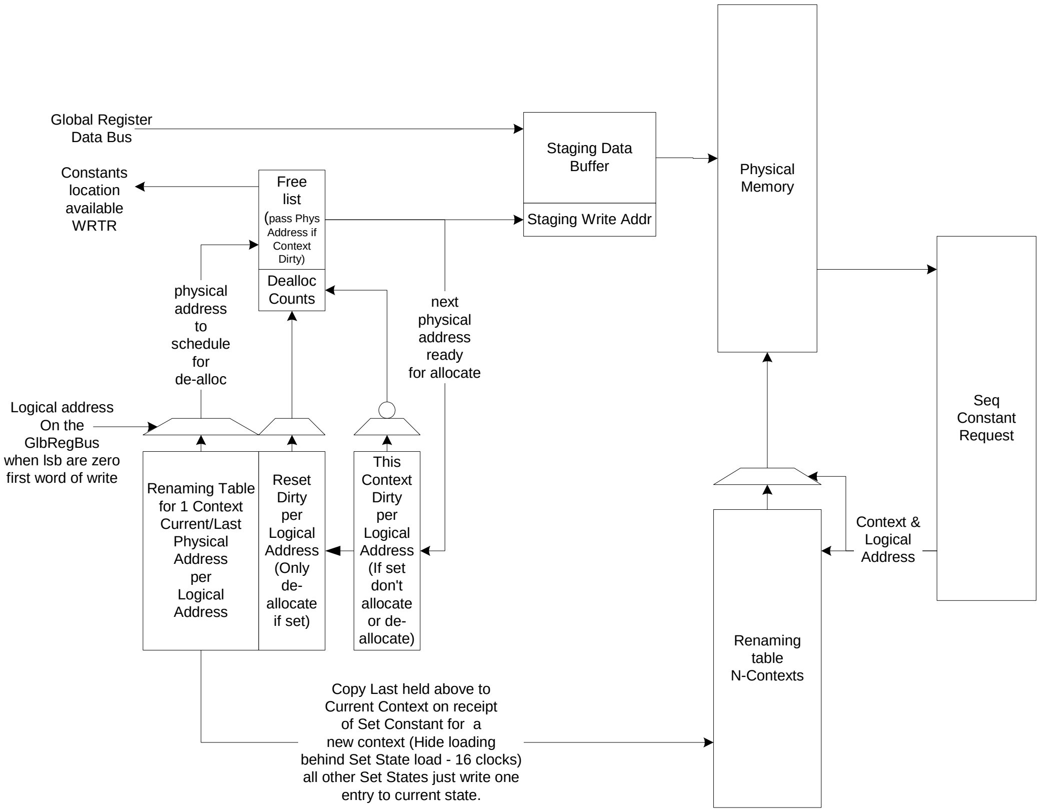

Global Register Data Bus
Staging Data Buffer
Staging Write Addr
Physical Memory
Constants location available WRTR
Free list (pass Phys Address if Context Dirty)
Dealloc Counts
physical address to schedule for de-alloc
next physical address ready for allocate
Seq Constant Request
Logical address On the GlbRegBus when lsb are zero first word of write
Renaming Table for 1 Context Current/Last Physical Address per Logical Address
Reset Dirty per Logical Address (Only de-allocate if set)
This Context Dirty per Logical Address (If set don't allocate or de-allocate)
Context & Logical Address
Renaming table N-Contexts
Copy Last held above to Current Context on receipt of Set Constant for a new context (Hide loading behind Set State load - 16 clocks) all other Set States just write one entry to current state.

Figure 7: Constant management

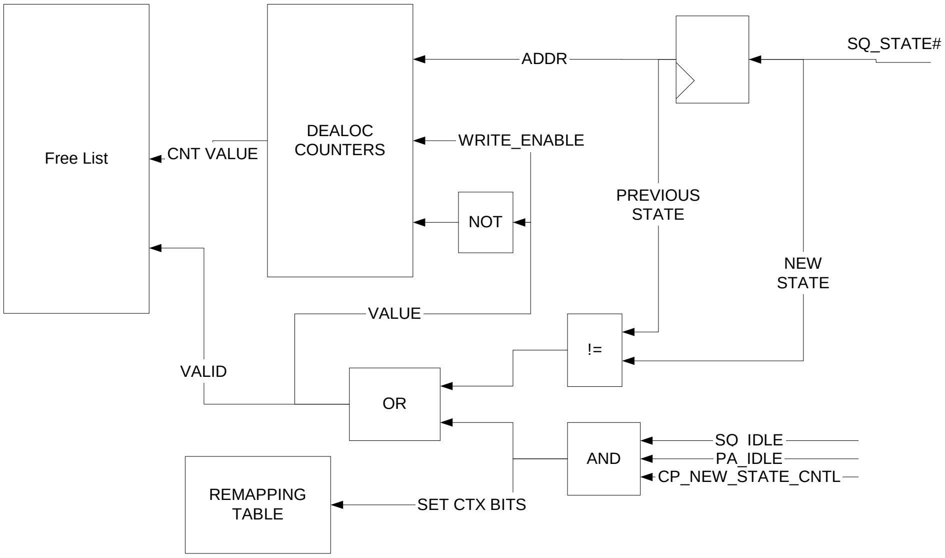


Figure 8: De-allocation mechanism for R400LE

5.3.3 *Dirty bits*

Two sets of dirty bits will be maintained per logical address. The first one will be set to zero on reset and set when the logical address is addressed. The second one will be set to zero whenever a new context is written and set for each address written while in this context. The reset dirty is not set, then writing to that logical address will not require de-allocation of whatever address stored in the renaming table. If it is set and the context dirty is not set, then the physical address store needs to be de-allocated and a new physical address is necessary to store the incoming data. If they are both set, then the data will be written into the physical address held in the renaming for the current logical address. No de-allocation or allocation takes place. This will happen when the driver does a set constant twice to the same logical address between context changes. NOTE: It is important to detect and prevent this, failure to do it will allow multiple writes to allocate all physical memory and thus hang because a context will not fit for rendering to start and thus free up space.

5.3.4 *Free List Block*

A free list block that would consist of a counter (called the IFC or Initial Free Counter) that would reset to zero and incremented every time a chunk of physical memory is used until they have all been used once. This counter would be checked each time a physical block is needed, and if the original ones have not been used up, us a new one, else check the free list for an available physical block address. The count is the physical address for when getting a chunk from the counter.

Storage of a free list big enough to store all physical block addresses.

Maintain three pointers for the free list that are reset to zero. The first one we will call write_ptr. This pointer will identify the next location to write the physical address of a block to be de-allocated. Note: we can never free more physical memory locations than we have. Once recording address the pointer will be incremented to walk the free list like a ring.

The second pointer will be called stop_ptr. The stop_ptr pointer will be advanced by the number of address chunks de-allocates when a context finishes. The address between the stop_ptr and write_ptr cannot be reused because they are still in use. But as soon as the context using then is dismissed the stop_ptr will be advanced.

The third pointer will be called read_ptr. This pointer will point will point to the next address that can be used for allocation as long as the read_ptr does not equal the stop_ptr and the IFC is at its maximum count.

5.3.5 *De-allocate Block*

This block will maintain a free physical address block count for each context. While in current context, a count shall be maintained specifying how many blocks were written into the free list at the write_ptr pointer. This count will be reset upon reset or when this context is active on the back and different than the previous context. It is actually a count of blocks in the previous context that will no longer be used. This count will be used to advance the write_ptr pointer to make available the set of physical blocks freed when the previous context was done. This allows the discard or de-allocation of any number of blocks in one clock.

5.3.6 *Operation of Incremental model*

The basic operation of the model would start with the write_ptr, stop_ptr, read_ptr pointers in the free list set to zero and the free list counter is set to zero. Also all the dirty bits and the previous context will be initialized to zero. When the first set constants happen, the reset dirty bit will not be set, so we will allocate a physical location from the free list counter because its not at the max value. The data will be written into physical address zero. Both the additional copy of the renaming table and the context zeros of the big renaming table will be updated for the logical address that was written by set start with physical address of 0. This process will be repeated for any logical address that are not dirty until the context changes. If a logical address is hit that has its dirty bits set while in the same context, both dirty bits would be set, so the new data will be over-written to the last physical address assigned for this logical address. When the first draw command of the context is detected, the previous context stored in the additional renaming table will be copied to the larger renaming table in the current (new) context location. Then the set constant logical address with be loaded with a new physical address during the copy and if the reset dirty was set, the physical address it replaced in the renaming table would be entered at the write_ptr pointer location on the free list and the write_ptr will be incremented. The de-allocation counter for the previous context (eight) will be incremented. This as set states come in for this context one of the following will happen:

1.) No dirty bits are set for the logical address being updated. A line will be allocated of the free-list counter or the free list at read_ptr pointer if read_ptr != to stop_ptr .
2.) Reset dirty set and Context dirty not set. A new physical address is allocated, the physical address in the renaming table is put on the free list at write_ptr and it is incremented along with the de-allocate counter for the last context.
3.) Context dirty is set then the data will be written into the physical address specified by the logical address.

This process will continue as long as set states arrive. This block will provide backpressure to the CP whenever he has not free list entries available (counter at max and stop_ptr == read_ptr). The command stream will keep a count of contexts of constants in use and prevent more than max constants contexts from being sent.

Whenever a draw packet arrives, the content of the re-mapping table is written to the correct re-mapping table for the context number. Also if the next context uses less constants than the current one all exceeding lines are moved to the free list to be de-allocated later. This happens in parallel with the writing of the re-mapping table to the correct memory.

Now preferable when the constant context leaves the last ALU clause it will be sent to this block and compared with the previous context that left. (Init to zero) If they differ than the older context will no longer be referenced and thus can be de-allocated in the physical memory. This is accomplished by adding the number of blocks freed this context to the stop_ptr pointer. This will make all the physical addresses used by this context available to the read_ptr allocate pointer for future allocation.

This device allows representation of multiple contexts of constants data with N copies of the logical address space. It also allows the second context to be represented as the first set plus some new additional data by just storing the delta's. It allows memory to be efficiently used and when the constants updates are small it can store multiple context. However, if the updates are large, less contexts will be stored and potentially performance will be degraded. Although it will still perform as well as a ring could in this case.

5.4 Constant Store Indexing

In order to do constant store indexing, the sequencer must be loaded first with the indexes (that come from the GPRs). There are 144 wires from the exit of the SP to the sequencer (9 bits pointers x 16 vertexes/clock). Since the data must pass thru the Shader pipe for the float to fixed conversion, there is a latency of 4 clocks (1 instruction)

between the time the sequencer is loaded and the time one can index into the constant store. The assembly will look like this

```
MOVA  R1.X,R2.X       // Loads the sequencer with the content of R2.X, also copies the content of R2.X into R1.X
NOP                   // latency of the float to fixed conversion
ADD   R3,R4,C0[R2.X]// Uses the state from the sequencer to add R4 to C0[R2.X] into R3
```

Note that we don't really care about what is in the brackets because we use the state from the MOVA instruction. R2.X is just written again for the sake of simplicity and coherency.

The storage needed in the sequencer in order to support this feature is 2*64*9 bits = 1152 bits.

5.5 Real Time Commands

The real time commands constants are written by the CP using the register mapped registers allocated for RT. It works is the same way than when dealing with regular constant loads BUT in this case the CP is not sending a logical address but rather a physical address and the reads are not passing thru the re-mapping table but are directly read from the memory. The boundary between the two zones is defined by the CONST_EO_RT control register. Similarly, for the fetch state, the boundary between the two zones is defined by the TSTATE_EO_RT control register.

5.6 Constant Waterfalling

In order to have a reasonable performance in the case of constant store indexing using the address register, we are going to have the possibility of using the physical memory port for read only. This way we can read 1 constant per clock and thus have a worst-case waterfall mode of 1 vertex per clock. There is a small synchronization issue related with this as we need for the SQ to make sure that the constants where actually written to memory (not only sent to the sequencer) before it can allow the first vector of pixels or vertices of the state to go thru the ALUs. To do so, the sequencer keeps 8 bits (one per render state) and sets the bits whenever the last render state is written to memory and clears the bit whenever a state is freed.

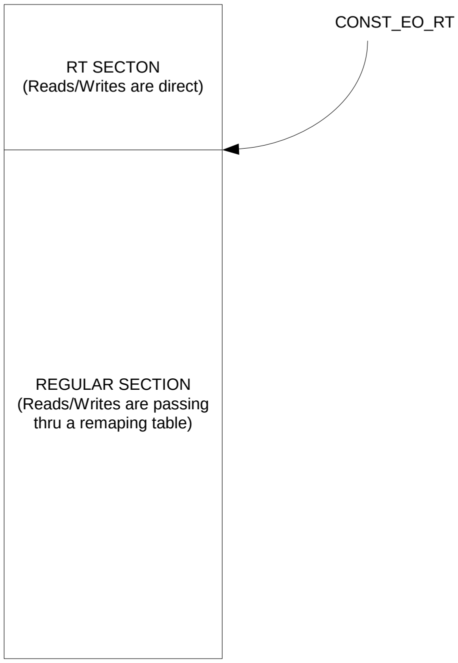


Figure 9: The Constant store

6. Looping and Branches

Loops and branches are planned to be supported and will have to be dealt with at the sequencer level. We plan on supporting constant loops and branches using a control program.

6.1 The controlling state.

The R400 controling state consists of:

Boolean[256:0]
Loop_count[7:0][31:0]
Loop_Start[7:0][31:0]
Loop_Step[7:0][31:0]

That is 256 Booleans and 32 loops.

We have a stack of 4 elements for nested calls of subroutines and 4 loop counters to allow for nested loops.

This state is available on a per shader program basis.

6.2 The Control Flow Program

We'd like to be able to code up a program of the form:

```
1:      Loop
2:      Exec    TexFetch
3:              TexFetch
4:              ALU
5:              ALU
6:              TexFetch
7:      End Loop
8:      ALU Export
```

But realize that 3: may be dependent on 2: and 4: is almost certainly dependent on 2: and 3:. Without clausing, these dependencies need to be expressed in the Control Flow instructions. Additionally, without separate 'texture clauses' and 'ALU clauses' we need to know which instructions to dispatch to the Texture Unit and which to the ALU unit. This information will be encapsulated in the flow control instructions.

Each control flow instruction will contain 2 bits of information for each (non-control flow) instruction:
a) ALU or Texture
b) Serialize Execution

(b) would force the thread to stop execution at this point (before the instruction is executed) and wait until all textures have been fetched. Given the allocation of reserved bits, this would mean that the count of an 'Exec' instruction would be limited to about 8 (non-control-flow) instructions. If more than this were needed, a second Exec (with the same conditions) would be issued.

Another function that relies upon 'clauses' is allocation and order of execution. We need to assure that pixels and vertices are exported in the correct order (even if not all execution is ordered) and that space in the output buffers are allocated in order. Additionally data can't be exported until space is allocated. A new control flow instruction:

Alloc <buffer select -- position,parameter, pixel or vertex memory. And the size required>.

would be created to mark where such allocation needs to be done. To assure allocation is done in order, the actual allocation for a given thread can not be performed unless the equivalent allocation for all previous threads is already completed. The implementation would also assure that execution of instruction(s) following the serialization due to the Alloc will occur in order -- at least until the next serialization or change from ALU to Texture. In most cases this will allow the exports to occur without any further synchronization. Only 'final' allocations or position allocations are

guaranteed to be ordered. Because strict ordering is required for pixels, parameters and positions, this implies only a single alloc for these structures. Vertex exports to memory do not require ordering during allocation and so multiple 'allocs' may be done.

6.2.1 *Control flow instructions table*

Here is the revised control flow instruction set.

Note that whenever a field is marked as RESERVED, it is assumed that all the bits of the field are cleared (0).

NOP		
47 … 44	43	42 … 0
0000	Addressing	RESERVED

This is a regular NOP.

Execute					
47 … 44	43	40 … 34	33 …16	15…12	11 … 0
0001	Addressing	RESERVED	Instructions type + serialize (9 instructions)	Count	Exec Address

Execute_End					
47 … 44	43	40 … 34	33 …16	15…12	11 … 0
0010	Addressing	RESERVED	Instructions type + serialize (9 instructions)	Count	Exec Address

Execute up to 9 instructions at the specified address in the instruction memory. The Instruction type field tells the sequencer the type of the instruction (LSB) (1 = Texture, 0 = ALU and whether to serialize or not the execution (MSB) (1 = Serialize, 0 = Non-Serialized). If Execute_End this is the last execution block of the shader program.

Conditional_Execute						
47 … 44	43	42	41 … 34	33…16	15 …12	11 … 0
0011	Addressing	Condition	Boolean address	Instructions type + serialize (9 instructions)	Count	Exec Address

Conditional_Execute_End						
47 … 44	43	42	41 … 34	33…16	15 …12	11 … 0
0100	Addressing	Condition	Boolean address	Instructions type + serialize (9 instructions)	Count	Exec Address

If the specified Boolean (8 bits can address 256 Booleans) meets the specified condition then execute the specified instructions (up to 9 instructions). If the condition is not met, we go on to the next control flow instruction. If Conditional_Execute_End and the condition is met, this is the last execution block of the shader program.

Conditional_Execute_Predicates							
47 … 44	43	42	41 … 36	35 … 34	33…16	15…12	11 … 0
0101	Addressing	Condition	RESERVED	Predicate vector	Instructions type + serialize (9 instructions)	Count	Exec Address

Conditional_Execute_Predicates_End							
47 … 44	43	42	41 … 36	35 … 34	33…16	15…12	11 … 0
0110	Addressing	Condition	RESERVED	Predicate vector	Instructions type + serialize (9 instructions)	Count	Exec Address

Check the AND/OR of all current predicate bits. If AND/OR matches the condition execute the specified number of instructions. We need to AND/OR this with the kill mask in order not to consider the pixels that aren’t valid. If the

condition is not met, we go on to the next control flow instruction. If Conditional_Execute_Predicates_End and the condition is met, this is the last execution block of the shader program.

Conditional_Execute_Predicates_No_Stall							
47 … 44	43	42	41 … 36	35 … 34	33…16	15…12	11 … 0
1101	Addressing	Condition	RESERVED	Predicate vector	Instructions type + serialize (9 instructions)	Count	Exec Address

Conditional_Execute_Predicates_No_Stall_End							
47 … 44	43	42	41 … 36	35 … 34	33…16	15…12	11 … 0
1110	Addressing	Condition	RESERVED	Predicate vector	Instructions type + serialize (9 instructions)	Count	Exec Address

Same as Conditionnal_Execute_Predicates but the SQ is not going to wait for the predicate vector to be updated. You can only set this in the compiler if you know that the predicate set is only a refinement of the current one (like a nested if) because the optimization would still work.

Loop_Start					
47 … 44	43	42 … 21	20 … 16	15…12	11 … 0
0111	Addressing	RESERVED	loop ID	RESERVED	Jump address

Loop Start. Compares the loop iterator with the end value. If loop condition not met jump to the address. Forward jump only. Also computes the index value. The loop id must match between the start to end, and also indicates which control flow constants should be used with the loop.

Loop_End						
47 … 44	43	42 … 24	23… 21	20 … 16	15…12	11 … 0
1000	Addressing	RESERVED	Predicate break	loop ID	RESERVED	start address

Loop end. Increments the counter by one, compares the loop count with the end value. If loop condition met, continue, else, jump BACK to the start of the loop. If predicate break != 0, then compares predicate vector n (specified by predicate break number). If all bits cleared then break the loop.

The way this is described does not prevent nested loops, and the inclusion of the loop id make this easy to do.

Conditionnal_Call						
47 … 44	43	42	41 … 34	33 … 13	12	11 … 0
1001	Addressing	Condition	Boolean address	RESERVED	Force Call	Jump address

If the condition is met, jumps to the specified address and pushes the control flow program counter on the stack. If force call is set the condition is ignored and the call is made always.

Return		
47 … 44	43	42 … 0
1010	Addressing	RESERVED

Pops the topmost address from the stack and jumps to that address. If nothing is on the stack, the program will just continue to the next instruction.

Conditionnal_Jump							
47 … 44	43	42	41… 34	33	32 … 13	12	11 … 0
1011	Addressing	Condition	Boolean address	FW only	RESERVED	Force Jump	Jump address

If force jump is set the condition is ignored and the jump is made always. If FW only is set then only forward jumps are allowed.

Allocate				
47 … 44	43	42…41	40 … 3~~4~~	2…~~3 …~~0
1100	Debug	Buffer Select	RESERVED	Size~~Allocation size~~

Buffer Select takes a value of the following:
01 – position export (ordered export)
10 – parameter cache or pixel export (ordered export)
11 – pass thru (out of order exports).

Size field is only used to reserve space in the export buffer for pass thru exports. Valid values are 1 (1 line) thru 9 (9 lines). It should be determined by the compiler/assembler by taking max index used +1.
~~Buffer Size takes a value of the following:~~
~~00 – 1 buffer~~
~~01 – 2 buffers~~
~~…~~
~~15 – 16 buffers~~

If debug is set this is a debug alloc (ignore if debug DB_ON register is set to off).

6.3 Implementation

The envisioned implementation has a buffer that maintains the state of each thread. A thread lives in a given location in the buffer during its entire life, but the buffer has FIFO qualities in that threads leave in the order that they enter. Actually two buffers are maintained -- one for Vertices and one for Pixels. The intended implementation would allow for:

16 entries for vertices
48 entries for pixels.

From each buffer, arbitration logic attempts to select 1 thread for the texture unit and 1 (interleaved) thread for the ALU unit. Once a thread is selected it is read out of the buffer, marked as invalid, and submitted to appropriate execution unit. It is returned to the buffer (at the same place) with its status updated once all possible sequential instructions have been executed. A switch from ALU to TEX or visa-versa or a Serialize_Execution modifier forces the thread to be returned to the buffer.

Each entry in the buffer will be stored across two physical pieces of memory - most bits will be stored in a 1 read port device. Only bits needed for thread arbitration will be stored in a highly multi-ported structure. The bits kept in the 1 read port device will be termed 'state'. The bits kept in the multi-read ported device will be termed 'status'.

'State Bits' needed include:

1. Control Flow Instruction Pointer (13 bits),
2. Execution Count Marker 4 bits),
3. Loop Iterators (4x9 bits),
4. Call return pointers (4x12 bits),
5. Predicate Bits (64 bits),
6. Export ID (1 bit),
7. Parameter Cache base Ptr (7 bits),
8. GPR Base Ptr (8 bits),
9. Context Ptr (3 bits).
10. LOD corrections (6x16 bits)
11. Valid bits (64 bits)
12. RT (1 bit) Signifies that this thread is a Real Time thread. This bit must be sent to the Constant store state machine when reading it.

Formatted: Bullets and Numbering

Absent from this list are 'Index' pointers. These are costly enough that I'm presuming that they are instead stored in the GPRs. The first seven fields above (Control Flow Ptr, Execution Count, Loop Counts, call return ptrs, Predicate bits, PC base ptr and export ID) are updated every time the thread is returned to the buffer based on how much progress has been mode on thread execution. GPR Base Ptr, Context Ptr and LOD corrections are unchanged throughout execution of the thread.

'Status Bits' needed include:

- Valid Thread
- Texture/ALU engine needed
- Texture Reads are outstanding
- Waiting on Texture Read to Complete
- Allocation Wait (2 bits)
- 00 – No allocation needed
- 01 – Position export allocation needed (ordered export)
- 10 – Parameter or pixel export needed (ordered export)
- 11 – pass thru (out of order export)
- Allocation Size (4 bits)
- Position Allocated
- First thread of a new context
- Event thread (NULL thread that needs to trickle down the pipe)
- Last (1 bit)
- Pulse SX (1 bit)

All of the above fields from all of the entries go into the arbitration circuitry. The arbitration circuitry will select a winner for both the Texture Engine and for the ALU engine. There are actually two sets of arbitration -- one for pixels and one for vertices. A final selection is then done between the two. But the rest of this implementation summary only considers the 'first' level selection which is similar for both pixels and vertices.

Texture arbitration requires no allocation or ordering so it is purely based on selecting the 'oldest' thread that requires the Texture Engine.

ALU arbitration is a little more complicated. First, only threads where either of Texture_Reads_outstanding or Waiting_on_Texture_Read_to_Complete are '0' are considered. Then if Allocation_Wait is active, these threads are further filtered based on whether space is available. If the allocation is position allocation, then the thread is only considered if all 'older' threads have already done their position allocation (position allocated bits set). If the allocation is parameter or pixel allocation, then the thread is only considered if it is the oldest thread. Also a thread is not considered if it is a parameter or pixel or position allocation, has its First_thread_of_a_new_context bit set and would cause ALU interleaving with another thread performing the same parameter or pixel or position allocation. Finally the 'oldest' of the threads that pass through the above filters is selected. If the thread needed to allocate, then at this time the allocation is done, based on Allocation_Size. If a thread has its “last” bit set, then it is also removed from the buffer, never to return.

If I now redefine 'clauses' to mean 'how many times the thread is removed from the thread buffer for the purpose of exection by either the ALU or Texture engine', then the minimum number of clauses needed is 2 -- one to perform the allocation for exports (execution automatically halts after an 'Alloc' instruction) (but doesn't performs the actual allocation) and one for the actual ALU/export instructions. As the 'Alloc' instruction could be part of a texture clause (presumably the final instruction in such a clause), a thread could still execute in this minimal number of 2 clauses, even if it involved texture fetching.

The Texture_Reads_Outstanding bit must be updated by the sequencer, based on keeping track of how many Texture Clauses have been executed by a given thread that have not yet had there data returned. Any number above 0 results in this bit being set. We could consider forcing synchronization such that two texture clauses for a given thread may not be outstanding at any time (that would be my preference for simplicity reasons and because it would require only very little change in the texture pipe interface). This would allow the sequencer to set the bit on execution of the texture clause, and allow the texture unit to return a pointer to the thread buffer on completion that clears the bit.

6.4 Data dependant predicate instructions

Data dependant conditionals will be supported in the R400. The only way we plan to support those is by supporting three vector/scalar predicate operations of the form:

PRED_SETE_# - similar to SETE except that the result is 'exported' to the sequencer.
PRED_SETNE_# - similar to SETNE except that the result is 'exported' to the sequencer.
PRED_SETGT_# - similar to SETGT except that the result is 'exported' to the sequencer
PRED_SETGTE_# - similar to SETGTE except that the result is 'exported' to the sequencer

For the scalar operations only we will also support the two following instructions:
PRED_SETE0_# – SETE0
PRED_SETE1_# – SETE1

The export is a single bit - 1 or 0 that is sent using the same data path as the MOVA instruction. The sequencer will maintain 4 sets of 64 bit predicate vectors (in fact 8 sets because we interleave two programs but only 4 will be exposed) and use it to control the write masking. This predicate is not maintained across clause boundaries. The # sign is used to specify which predicate set you want to use 0 thru 3.

Then we have two conditional execute bits. The first bit is a conditional execute “on” bit and the second bit tells us if we execute on 1 or 0. For example, the instruction:

P0_ADD_# R0,R1,R2

Is only going to write the result of the ADD into those GPRs whose predicate bit is 0. Alternatively, P1_ADD_# would only write the results to the GPRs whose predicate bit is set. The use of the P0 or P1 without precharging the sequencer with a PRED instruction is undefined.

{Issue: do we have to have a NOP between PRED and the first instruction that uses a predicate?}

6.5 HW Detection of PV,PS

Because of the control program, the compiler cannot detect statically dependant instructions. In the case of non-masked writes and subsequent reads the sequencer will insert uses of PV,PS as needed. This will be done by comparing the read address and the write address of consecutive instructions. For masked writes, the sequencer will insert NOPs wherever there is a dependant read/write.

The sequencer will also have to insert NOPs between PRED_SET and MOVA instructions and their uses.

6.6 Register file indexing

Because we can have loops in fetch clause, we need to be able to index into the register file in order to retrieve the data created in a fetch clause loop and use it into an ALU clause. The instruction will include the base address for register indexing and the instruction will contain these controls:

Bit7	Bit 6	
0	0	'absolute register'
0	1	'relative register'
1	0	'previous vector'
1	1	'previous scalar'

In the case of an absolute register we just take the address as is. In the case of a relative register read we take the base address and we add to it the loop_index and this becomes our new address that we give to the shader pipe.

The sequencer is going to keep a loop index computed as such:

Index = Loop_iterator*Loop_step + Loop_start.

We loop until loop_iterator = loop_count. Loop_step is a signed value [-128…127]. The computed index value is a 10 bit counter that is also signed. Its real range is [-256,256]. The tenth bit is only there so that we can provide an out of range value to the “indexing logic” so that it knows when the provided index is out of range and thus can make the necessary arrangements.

6.7 Debugging the Shaders

In order to be able to debug the pixel/vertex shaders efficiently, we provide 2 methods.

6.7.1 *Method 1: Debugging registers*

Current plans are to expose 2 debugging, or error notification, registers:
1. address register where the first error occurred
2. count of the number of errors

The sequencer will detect the following groups of errors:
- count overflow
- constant indexing overflow
- register indexing overflow

Compiler recognizable errors:
- jump errors
  relative jump address > size of the control flow program
- call stack
  call with stack full
  return with stack empty

A jump error will always cause the program to break. In this case, a break means that a clause will halt execution, but allowing further clauses to be executed.

With all the other errors, program can continue to run, potentially to worst-case limits. The program will only break if the DB_PROB_BREAK register is set.

If indexing outside of the constant or the register range, causing an overflow error, the hardware is specified to return the value with an index of 0. This could be exploited to generate error tokens, by reserving and initializing the 0th register (or constant) for errors.

{ISSUE : Interrupt to the driver or not?}

6.7.2 *Method 2: Exporting the values in the GPRs*

1) The sequencer will have a debug active, count register and an address register for this mode.

Under the normal mode execution follows the normal course.

Under the debug mode it is assumed that the program is always exporting n debug vectors and that all other exports to the SX block (position, color, z, ect) will been turned off (changed into NOPs) by the sequencer (even if they occur before the address stated by the ADDR debug register).

7. Pixel Kill Mask

A vector of 64 bits is kept by the sequencer per group of pixels/vertices. Its purpose is to optimize the texture fetch requests and allow the shader pipe to kill pixels using the following instructions:

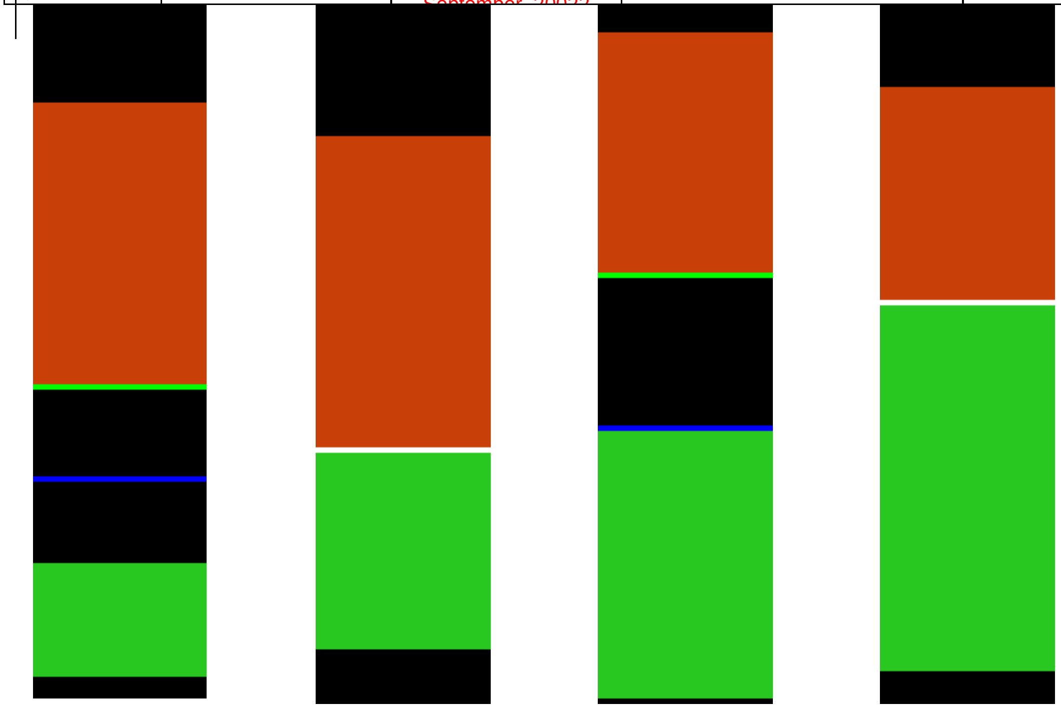
MASK_SETE
MASK_SETNE
MASK_SETGT
MASK_SETGTE

8. Multipass vertex shaders (HOS)

Multipass vertex shaders are able to export from the 6 last clauses but to memory ONLY.

9. Register file allocation

The register file allocation for vertices and pixels can either be static or dynamic. In both cases, the register file in managed using two round robins (one for pixels and one for vertices). In the dynamic case the boundary between pixels and vertices is allowed to move, in the static case it is fixed to 128-VERTEX_REG_SIZE for vertices and PIXEL_REG_SIZE for pixels.



Above is an example of how the algorithm works. Vertices come in from top to bottom; pixels come in from bottom to top. Vertices are in orange and pixels in green. The blue line is the tail of the vertices and the green line is the tail of the pixels. Thus anything between the two lines is shared. When pixels meets vertices the line turns white and the boundary is static until both vertices and pixels share the same "unallocated bubble". Then the boundary is allowed to move again. The numbering of the GPRs starts from the bottom of the picture at index 0 and goes up to the top at index 127.

10. Fetch Arbitration

The fetch arbitration logic chooses one of the n potentially pending fetch clauses to be executed. The choice is made by looking at the Vs and Ps reservation stations and picking the first one ready to execute. Once chosen, the clause state machine will send one 2x2 fetch per clock (or 4 fetches in one clock every 4 clocks) until all the fetch instructions of the clause are sent. This means that there cannot be any dependencies between two fetches of the same clause.

The arbitrator will not wait for the fetches to return prior to selecting another clause for execution. The fetch pipe will be able to handle up to X(?) in flight fetches and thus there can be a fair number of active clauses waiting for their fetch return data.

11. ALU Arbitration

ALU arbitration proceeds in almost the same way than fetch arbitration. The ALU arbitration logic chooses one of the n potentially pending ALU clauses to be executed. The choice is made by looking at the Vs and Ps reservation stations and picking the first one ready to execute. There are two ALU arbiters, one for the even clocks and one for the odd clocks. For example, here is the sequencing of two interleaved ALU clauses (E and O stands for Even and Odd sets of 4 clocks):

Einst0 Oinst0 Einst1 Oinst1 Einst2 Oinst2 Einst0 Oinst3 Einst1 Oinst4 Einst2 Oinst0…

Proceeding this way hides the latency of 8 clocks of the ALUs. Also note that the interleaving also occurs across clause boundaries.

12. Handling Stalls

When the output file is full, the sequencer prevents the ALU arbitration logic from selecting the last clause (this way nothing can exit the shader pipe until there is place in the output file. If the packet is a vertex packet and the position buffer is full (POS_FULL) then the sequencer also prevents a thread from entering an exporting clause. The sequencer will set the OUT_FILE_FULL signal n clocks before the output file is actually full and thus the ALU arbiter will be able read this signal and act accordingly by not preventing exporting clauses to proceed.

13. Content of the reservation station FIFOs

The reservation FIFOs contain the state of the vector of pixels and vertices. We have two sets of those: one for pixels, and one for vertices. They contain 3 bits of Render State 7 bits for the base address of the GPRs, some bits for LOD correction and coverage mask information in order to fetch fetch for only valid pixels, the quad address.

14. The Output File

The output file is where pixels are put before they go to the RBs. The write BW to this store is 256 bits/clock. Just before this output file are staging registers with write BW 512 bits/clock and read BW 256 bits/clock. The staging registers are 4x128 (and there are 16 of those on the whole chip).

15. IJ Format

The IJ information sent by the PA is of this format on a per quad basis:

We have a vector of IJ's (one IJ per pixel at the centroid of the fragment or at the center of the pixel depending on the mode bit). All pixel's parameters are always interpolated at full 20x24 mantissa precision.

$$P0 = A + I(0)*(B - A) + J(0)*(C - A)$$
$$P1 = A + I(1)*(B - A) + J(1)*(C - A)$$
$$P2 = A + I(2)*(B - A) + J(2)*(C - A)$$
$$P3 = A + I(3)*(B - A) + J(3)*(C - A)$$

P0	P1
P2	P3

Multiplies (Full Precision): 8
Subtracts 19x24 (Parameters): 2
Adds: 8

FORMAT OF P's IJ : Mantissa 20 Exp 4 for I + Sign
Mantissa 20 Exp 4 for J + Sign

Total number of bits : 20*8 + 4*8 + 4*2 = 200.

All numbers are kept using the un-normalized floating point convention: if exponent is different than 0 the number is normalized if not, then the number is un-normalized. The maximum range for the IJs (Full precision) is +/- 1024.

15.1 Interpolation of constant attributes

Because of the floating point imprecision, we need to take special provisions if all the interpolated terms are the same or if two of the terms are the same.

16. Staging Registers

In order for the reuse of the vertices to be 14, the sequencer will have to re-order the data sent IN ORDER by the VGT for it to be aligned with the parameter cache memory arrangement. Given the following group of vertices sent by the VGT:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 || 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 || 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 || 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63

The sequencer will re-arrange them in this fashion:

0 1 2 3 16 17 18 19 32 33 34 35 48 49 50 51 || 4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55 || 8 9 10 11 24 25 26 27 40 41 42 43 56 57 58 59 || 12 13 14 15 28 29 30 31 44 45 46 47 60 61 62 63

The || markers show the SP divisions. In the event a shader pipe is broken, the SQ is responsible to insert padding to account for the missing pipe. For example, if SP1 is broken, vertices 4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55 will not be sent by the VGT to the SQ **AND** the SQ is responsible to "jump" over these vertices in order for no valid vertices to be sent to an invalid SP.

The most straightforward, *non-compressed* interface method would be to convert, in the VGT, the data to 32-bit floating point prior to transmission to the VSISRs. In this scenario, the data would be transmitted to (and stored in) the VSISRs in full 32-bit floating point. This method requires three 24-bit fixed-to-float converters in the VGT. Unfortunately, it also requires and additional 3,072 bits of storage across the VSISRs. This interface is illustrated in Figure 11~~Figure 11Figure 11~~. The area of the fixed-to-float converters and the VSISRs for this method is roughly estimated as 0.759sqmm using the R300 process. The gate count estimate is shown in Figure 10~~Figure 10Figure 10~~.

Basis for 8-deep Latch Memory (from R300)

8x24-bit	11631 μ^2	60.57813 μ^2 per bit
Area of 96x8-deep Latch Memory	46524 μ^2	
Area of 24-bit Fix-to-float Converter	4712 μ^2 per converter	

Method 1	Block	Quantity	Area
	F2F	3	14136
	8x96 Latch	16	744384
			758520 μ^2

Figure 10:Area Estimate for VGT to Shader Interface

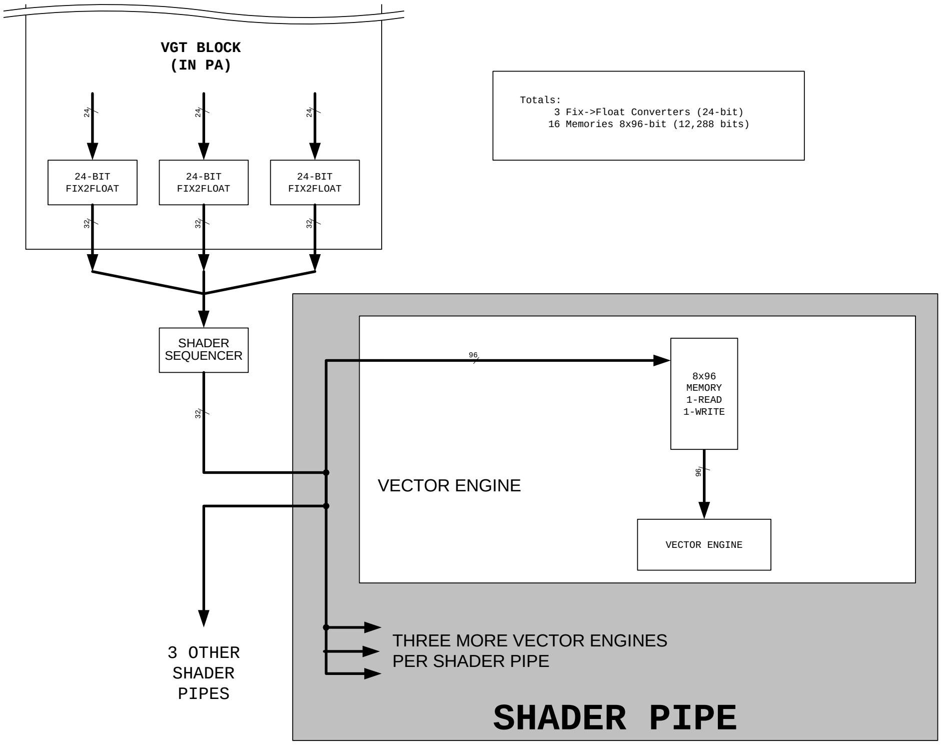


Figure 11:VGT to Shader Interface

17. The parameter cache

The parameter cache is where the vertex shaders export their data. It consists of 16 128x128 memories (1R/1W). The reuse engine will make it so that all vertexes of a given primitive will hit different memories. The allocation method for these memories is a simple round robin. The parameter cache pointers are mapped in the following way: 4MSBs are the memory number and the 7 LSBs are the address within this memory.

MEMORY NUMBER 4 bits	ADDRESS 7 bits

The PA generates the parameter cache addresses as the positions come from the SQ. All it needs to do is keep a Current_Location pointer (7 bits only) and as the positions comes increment the memory number. When the memory number field wraps around, the PA increments the Current_Location by VS_EXPORT_COUNT (a snooped register from the SQ). As an example, say the memories are all empty to begin with and the vertex shader is exporting 8 parameters per vertex (VS_EXPORT_COUNT = 8). The first position received is going to have the PC address 00000000000 the second one 00010000000, third one 00100000000 and so on up to 11110000000. Then the next position received (the 17th) is going to have the address 00000001000, the 18th 00010001000, the 19th 00100001000 and so on. The Current_location is NEVER reset BUT on chip resets. The only thing to be careful about is that if the SX doesn't send you a full group of positions (<64) then you need to fill the address space so that the next group starts correctly aligned (for example if you receive only 33 positions then you need to add 2*VS_EXPORT_COUNT to Current_Location and reset the memory count to 0 before the next vector begins).

17.1 Export restrictions

17.1.1 *Pixel exports:*

Pixels can export 1,2,3 or 4 color buffers to the SX(+z). The exports will be done in order. The PRED_OPTIMIZE function has to be turned of if the exports are done using interleaved predicated instructions. The exports will always be ordered to the SX.

17.1.2 *Vertex exports:*

Position or parameter caches can be exported in any order in the shader program. It is always better to export posistion as soon as possible. Position has to be exported in a single export block (no texture instructions can be placed between the exports). Parameter cache exports can be done in any order with texture instructions interleaved. The PRED_OPTIMIZE function has to be turned of if the exports are done using interleaved predicated instructions to the Parameter cache (see Arbitration restrictions for details). The exports will always be allocated in order to the SX.

17.1.3 *Pass thru exports:*

Pass thru exports have to be done in groups of the form:

```
Alloc 4 (8 or 12)
Execute ALU(ADDR) ALU(DATA) ALU(DATA) ALU(DATA)…
```

They cannot have texture instructions interleaved in the export block. These exports **are not guaranteed to be ordered**.

Also, when doing a pass thru export, Position MUST be exported AFTER all pass thru exports. This position export is used to synchronize the chip when doing a transition from pass thru shader to regular shader and vice versa.

17.2 Arbitration restrictions

Here are the Sequencer arbitration restrictions:

1) Cannot execute a serialized thread if the corresponding texture pending bit is set
2) Cannot allocate position if any older thread has not allocated position
3) If last thread is marked as not valid AND marked as last and we are about to execute the second to oldest thread also marked last then:
    a. Both threads must be from the same context (cannot allow a first thread)
    b. Must turn off the predicate optimization for the second thread
4) Cannot execute a texture clause if texture reads are pending
5) Cannot execute last if texture pending (even if not serial)

18. Export Types

The export type (or the location where the data should be put) is specified using the destination address field in the ALU instruction. Here is a list of all possible export modes:

18.1 Vertex Shading

0:15	- 16 parameter cache
16:31	- Empty (Reserved?)
32	- Export Address
33:~~40~~ 41	- ~~8~~ 9 vertex exports to the frame buffer and index
~~41~~42:47	- Empty
48:55	- 8 debug export (interpret as normal vertex export)
60	- export addressing mode
61	- Empty
62	- position

63 - sprite size export that goes with position export (~~point_h,point_w,edgeflag,misc)~~X= point size, Y= edge flag is bit 0, Z= VtxKill is bitwise OR of bits 30:0. Any bit other than sign means VtxKill.)

18.2 Pixel Shading

0	- Color for buffer 0 (primary)
1	- Color for buffer 1
2	- Color for buffer 2
3	- Color for buffer 3
4:~~7~~15	- Empty
~~8~~16	- Buffer 0 Color/Fog (primary)
~~9~~17	- Buffer 1 Color/Fog
1~~0~~8	- Buffer 2 Color/Fog
1~~1~~9	- Buffer 3 Color/Fog
~~12~~20:~~15~~31	- Empty
~~16:31~~	~~- Empty (Reserved?)~~
32	- Export Address
33:~~40~~41	- ~~8~~ 9 exports for multipass pixel shaders.
4~~1~~2:47	- Empty
48:55	- 8 debug exports (interpret as normal pixel export)
60 ~~60~~	- export addressing mode
~~60~~61	- Z for primary buffer (Z exported to 'alpha' component)
~~61~~62:6~~2~~3	- Empty
~~63~~	~~- Z for primary buffer (Z exported to 'alpha' component)~~

Formatted: Bullets and Numbering

19. Special Interpolation modes

19.1 Real time commands

We are unable to use the parameter memory since there is no way for a command stream to write into it. Instead we need to add three 16x128 memories (one for each of three vertices x 16 interpolants). These will be mapped onto the register bus and written by type 0 packets, and output to the the parameter busses (the sequencer and/or PA need to be able to address the realtime parameter memory as well as the regular parameter store. For higher performance we should be able able to view them as two banks of 16 and do double buffering allowing one to be loaded, while the other is rasterized with. Most overlay shaders will need 2 or 4 scalar coordinates, one option might be to restrict the memory to 16x64 or 32x64 allowing only two interpolated scalars per cycle, the only problem I see with this is, if we view support for 16 vector-4 interpolants important (true only if we map Microsoft's high priority stream to the realtime stream), then the PA/sequencer need to support a realtime-specific mode where we need to address 32 vectors of parameters instead of 16. This mode is triggered by the primitive type: REAL TIME. The actual memories are in the in the SX blocks. The parameter data memories are hooked on the RBBM bus and are loaded by the CP using register mapped memory.

19.2 Sprites/ XY screen coordinates/ FB information

XY screen coordinates may be needed in the shader program. This functionality is controlled by the param_gen_I0 register (in SQ) in conjunction with the SND_XY register (in SC) and the param_gen_pos. Also it is possible to send the faceness information (for OGL front/back special operations) to the shader using the same control register. Here is a list of all the modes and how they interact together:

The Data is going to be written in the register specified by the param_gen_pos register.

Param_Gen_I0 disable, snd_xy disable = No modification
Param_Gen_I0 disable, snd_xy enable = No modification
Param_Gen_I0 enable, snd_xy disable = Sign(faceness)garbage,(Sign Point)garbage,Sign(Line)s, t
Param_Gen_I0 enable, snd_xy enable = Sign(faceness)screenX,(Sign Point)screenY,Sign(Line)s, t

In other words,

The generated vector is (X in RED, Y in GREEN, S in BLUE and T in ALPHA):
X,Y,S,T
These values are always supposed to be positive and any shader use of them should use the ABS function (as their sign bits will now be used for flags).
SignX = BackFacing
SignY = Point Primitive
SignS = Line Primitive
SignT = currently unused as a flag.

If !Point & !Line, then it is a Poly.

I would assume that one implementation which allows for generic texture lookup (using 3D maps) for poly stipple and AA for the driver would be

```
if(Y<0) {
        R = 0.0 (Point)
} else if (S < 0) {
        R = 1.0 (Line)
} else {
        R = 2.0 (Poly)
}
```

19.3 Auto generated counters

In the cases we are dealing with multipass shaders, the sequencer is going to generate a vector count to be able to both use this count to write the 1st pass data to memory and then use the count to retrieve the data on the 2nd pass. The count is always generated in the same way but it is passed to the shader in a slightly different way depending on the shader type (pixel or vertex). This is toggled on and off using the GEN_INDEX_PIX/VTX register. The sequencer is going to keep two counters, one for pixels and one for vertices. Every time a full vector of vertices or pixels is written to the GPRs the counter is incremented. Every time a ~~state change is detected~~RST_PIX_COUNT or RST_VTX_COUNT events are received, the corresponding counter is reset. While there is only one count broadcast to the GPRs, the LSB are hardwired to specific values making the index different for all elements in the vector. Since the count must be different for all pixels/vertices and the 4 LSBs (16 positions) are hardwired to the corresponding shader unit the SQ has two choices:

1) Maintain a 19 bit counter that counts the vectors of 64. In this case the phase must be appended to the count before the count is broadcast to the SPs:

Formatted: Bullets and Numbering

Counter (19 bits)	Phase (2 bits)	Hardwired (4 bits)

2) Maintain a 21 bits counter that counts sub-vectors of 16. In this case only the counter is sent to the Sps:

Formatted: Bullets and Numbering

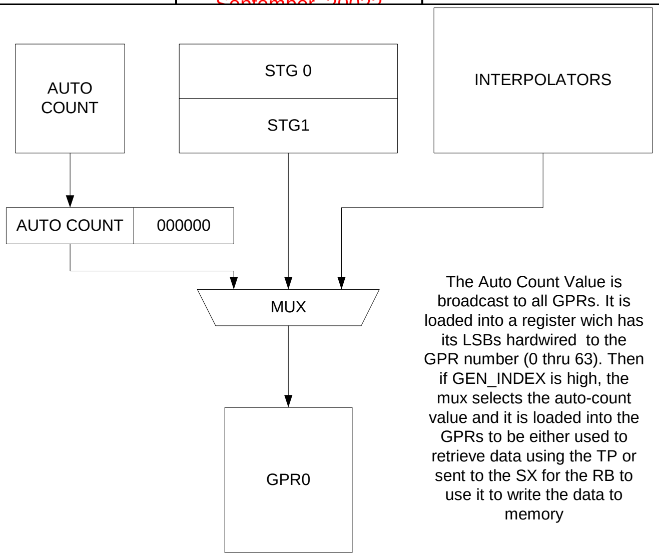
Counter (21 bits)	Harwired (4 bits)

19.3.1 *Vertex shaders*

In the case of vertex shaders, if GEN_INDEX_VTX is set, the data will be put into the x field of the third register (it means that the compiler must allocate 3 GPRs in all multipass vertex shader modes).

19.3.2 *Pixel shaders*

In the case of pixel shaders, if GEN_INDEX_PIX is set, the data will be put in the x field of the param_gen_pos+1 register.



The Auto Count Value is broadcast to all GPRs. It is loaded into a register wich has its LSBs hardwired to the GPR number (0 thru 63). Then if GEN_INDEX is high, the mux selects the auto-count value and it is loaded into the GPRs to be either used to retrieve data using the TP or sent to the SX for the RB to use it to write the data to memory

Figure 12: GPR input mux Control

20. State management

Every clock, the sequencer will report to the CP the oldest states still in the pipe. These are the states of the programs as they enter the last ALU clause.

20.1 Parameter cache synchronization

In order for the sequencer not to begin a group of pixels before the associated group of vertices has finished, the sequencer will keep a 6 bit count per state (for a total of 8 counters). These counters are initialized to 0 and every time a vertex shader exports its data TO THE PARAMETER CACHE, the corresponding pointer is incremented. When the SC sends a new vector of pixels with the SC_SQ_new_vector bit asserted, the sequencer will first check if the count is greater than 0 before accepting the transmission (it will in fact accept the transmission but then lower its ready to receive). Then the sequencer waits for the count to go to one and decrements it. The sequencer can then issue the group of pixels to the interpolators. Every time the state changes, the new state counter is initialized to 0.

21. XY Address imports

The SC will be able to send the XY addresses to the GPRs. It does so by interleaving the writes of the IJs (to the IJ buffer) with XY writes (to the XY buffer). Then when writing the data to the GPRs, the sequencer is going to interpolate the IJ data or pass the XY data thru a Fix→float converter and expander and write the converted values to the GPRs. The Xys are currently SCREEN SPACE COORDINATES. The values in the XY buffers will wrap. See section 19.2 for details on how to control the interpolation in this mode.

21.1 Vertex indexes imports

In order to import vertex indexes, we have 16 8x96 staging registers. These are loaded one line at a time by the VGT block (96 bits). They are loaded in floating point format and can be transferred in 4 or 8 clocks to the GPRs.

22. Registers

Please see the auto-generated web pages for register definitions.

23. Interfaces

23.1 External Interfaces

Whenever an x is used, it means that the bus is broadcast to all units of the same name. For example, if a bus is named SQ→SPx it means that SQ is going to broadcast the same information to all SP instances.

23.2 SC to SP Interfaces

23.2.1 *SC_SP#*

There is one of these interfaces at front of each of the SP (buffer to stage pixel interpolators). This interface transmits the I,J data for pixel interpolation. For the entire system, two quads per clock are transferred to the 4 SPs, so each of these 4 interfaces transmits one half of a quad per clock. The interface below describes a half of a quad worth of data.
The actual data which is transferred per quad is
Ref Pix I => S4.20 Floating Point I value *4
Ref Pix J => S4.20 Floating Point J value *4

This equates to a total of 200 bits which transferred over 2 clocks
and therefor needs an interface 100 bits wide

Additionally, X,Y data (12-bit unsigned fixed) is conditionally sent across this data bus over the same wires in an additional clock. The X,Y data is sent on the lower 24 bits of the data bus with faceness in the msb.
Transfers across these interfaces are synchronized with the SC_SQ IJ Control Bus transfers.

The data transfer across each of these busses is controlled by a IJ_BUF_INUSE_COUNT in the SC. Each time the SC has sent a pixel vector's worth of data to the SPs, he will increment the IJ_BUF_INUSE_COUNT count. Prior to sending the next pixel vectors data, he will check to make sure the count is less than MAX_BUFER_MINUS_2, if not the SC will stall until the SQ returns a pipelined pulse to decrement the count when he has scheduled a buffer free. Note: We could/may optimize for the case of only sending only IJ to use all the buffers to pre-load more. Currently it is planned for the SP to hold 2 double buffers of I,J data and two buffers of X,Y data, so if either X,Y or Centers and Centroids are on, then the SC can send two Buffers.

In at least the initial version, the SC shall send 16 quads per pixel vector even if the vector is not full. This will increment buffer write address pointers correctly all the time. (We may revisit this for both the SX,SP,SQ and add a EndOfVector signal on all interfaces to quit early. We opted for the simple mode first with a belief that only the end of packet and multiple new vector signals should cause a partial vector and that this would not really be significant performance hit.)

Name	Bits	Description
SC_SP#_data	100	IJ information sent over 2 clocks (or X,Y in 24 LSBs with faceness in upper bit) **Type 0 or 1**, First clock I, second clk J Field ULC URC LLC LRC Bits [63:39] [38:26] [25:13] [12:0] Format SE4M20 SE4M20 SE4M20 SE4M20 **Type 2** Field Face X Y Bits [~~63~~24] [23:12] [11:0] Format Bit Unsigned Unsigned
SC_SP#_valid	1	Valid
SC_SP#_last_quad_data	1	This bit will be set on the last transfer of data per quad.
SC_SP#_type	2	0 -> Indicates centroids 1 -> Indicates centers 2 -> Indicates X,Y Data and faceness on data bus The SC shall look at state data to determine how many types to send for the

		interpolation process.

The # is included for clarity in the spec and will be replaced with a prefix of u#_ in the verilog module statement for the SC and the SP block will have neither because the instantiation will insert the prefix.

23.2.2 *SC_SQ*

This is the control information sent to the sequencer in order to synchronize and control the interpolation and/or loading data into the GPRs needed to execute a shader program on the sent pixels. This data will be sent over two clocks per transfer with 1 to 16 transfers. Therefore the bus (approx ~~94~~ 108 bits) could be folded in half to approx ~~49~~ 54 bits.

Name	Bits	Description
SC_SQ_data	46	Control Data sent to the SQ 1 clk transfers Event – valid data consist of event_id and state_id. Instruct SQ to post an event vector to send state id and event_id through request fifo and onto the reservation stations making sure state id and/or event_id gets back to the CP. Events only follow end of packets so no pixel vectors will be in progress. Empty Quad Mask – Transfer Control data consisting of pc_dealloc or new_vector. Receipt of this is to transfer pc_dealloc or new_vector without any valid quad data. New vector will always be posted to request fifo and pc_dealloc will be attached to any pixel vector outstanding or posted in request fifo if no valid quad outstanding. 2 clk transfers Quad Data Valid – Sending quad data with or without new_vector or pc_dealloc. New vector will be posted to request fifo with or without a pixel vector and pc_dealloc will be posted with a pixel vector unless none is in progress. In this case the pc_dealloc will be posted in the request queue. Filler quads will be transferred with The Quad mask set but the pixel corresponding pixel mask set to zero.
SC_SQ_valid	1	SC sending valid data, 2nd clk could be all zeroes

SC_SQ_data – first clock and second clock transfers are shown in the table below.

Name	BitField	Bits	Description
1st Clock Transfer			
SC_SQ_event	0	1	This transfer is a 1 clock event vector Force quad_mask = new_vector=pc_dealloc=0
SC_SQ_event_id	[4:1]	4	This field identifies the event 0 => denotes an End Of State Event 1

			=> TBD
~~SC_SQ_pc_dealloc~~SC_SQ_state_id	[7:5]~~[7:5]~~	3~~3~~	~~Deallocation token for the Parameter Cache~~State/constant pointer (6*3+3)
SC_SQ_pc_dealloc	[10:8]	3	Deallocation token for the Parameter Cache
SC_SQ_new_vector	11~~8~~	1	The SQ must wait for Vertex shader done count > 0 and after dispatching the Pixel Vector the SQ will decrement the count.
SC_SQ_quad_mask	[1~~2~~5:12~~9~~]	4	Quad Write mask left to right SP0 => SP3
SC_SQ_end_of_prim	1~~3~~6	1	End Of the primitive
SC_SQ_pix_mask	[32:17]	16	Valid bits for all pixels SP0=>SP3 (UL,UR,LL,LR)
SC_SQ_provok_vtx	[3~~7~~4:3~~6~~3]	2	Provoking vertex for flat shading
~~SC_SQ_pc_ptr0~~SC_SQ_lod_correct_0	[4~~8~~3:3~~8~~5]	~~11~~9	~~Parameter Cache pointer for vertex 0~~LOD correction for quad 0 (SP0) (9 bits per quad)
SC_SQ_lod_correct_1	[52:44]	9	LOD correction for quad 1 (SP1) (9 bits per quad)
2nd Clock Transfer			
SC_SQ_lod_correct_2~~SC_SQ_pc_ptr1~~	[8:0]~~[10:0]~~	9~~11~~	LOD correction for quad 2 (SP2) (9 bits per quad)~~Parameter Cache pointer for vertex 1~~
SC_SQ_lod_correct_3	[17:9]	9	LOD correction for quad 3 (SP3) (9 bits per quad)
SC_SQ_pc_ptr0	[28:18]	11	Parameter Cache pointer for vertex 0
SC_SQ_pc_ptr~~2~~1	[~~21~~39:~~11~~29]	11	Parameter Cache pointer for vertex 1~~2~~
SC_SQ_pc_ptr2 ~~SC_SQ_lod_correct~~	[~~45~~50:~~22~~40]	~~24~~11	Parameter Cache pointer for vertex 2~~LOD correction per quad (6 bits per quad)~~
SC_SQ_prim_type	[~~48~~53:~~46~~51]	3	Stippled line and Real time command need to load tex cords from alternate buffer 000: Sprite (point) 001: Line 010: Tri_rect 100: Realtime Sprite (point) 101: Realtime Line 110: Realtime Tri_rect

Name	**Bits**	**Description**
SQ_SC_free_buff	1	Pipelined bit that instructs SC to decrement count of buffers in use.
SQ_SC_dec_cntr_cnt	1	Pipelined bit that instructs SC to decrement count of new vector and/or event sent to prevent SC from overflowing SQ interpolator/Reservation request fifo.

The scan converter will submit a partial vector whenever:

1.) He gets a primitive marked with an end of packet signal.

2.) A current pixel vector is being assembled with at least one or more valid quads and the vector has been marked for deallocate when a primitive marked new_vector arrives. The Scan Converter will submit a partial vector (up to 16quads with zero pixel mask to fill out the vector) prior to submitting the new_vector marker\primitive.

(This will prevent a hang which can be demonstrated when all primitives in a packet three vectors are culled except for a one quad primitive that gets marked pc_dealloc (vertices maximum size). In this case two new_vectors are submitted and processed, but then one valid quad with the pc_dealloc creates a vector and then the new would wait for another vertex vector to be processed, but the one being waited for could never export until the pc_dealloc signal made it through and thus the hang.)

23.2.3 SQ to SX(SP): Interpolator bus

Name	Direction	Bits	Description
SQ_SPx_interp_flat_vtx	SQ→SPx	2	Provoking vertex for flat shading
SQ_SPx_interp_flat_gouraud	SQ→SPx	1	Flat or gouraud shading
SQ_SPx_interp_cyl_wrap	SQ→SPx	4	Wich channel needs to be cylindrical wrapped
SQ_SPx_interp_param_gen	SQ→SPx	1	Generate Parameter
SQ_SPx_interp_prim_type	SQ→SPx	2	Bits [1:0] of primitive type sent by SC
SQ_SPx_interp_buff_swap	SQ→SPx	1	Swapp IJ buffers
SQ_SPx_interp_IJ_line	SQ→SPx	2	IJ line number
SQ_SPx_interp_mode	SQ→SPx	1	Center/Centroid sampling
SQ_SXx_pc_ptr0	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_pc_ptr1	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_pc_ptr2	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_rt_sel	SQ→SXx	1	Selects between RT and Normal data (Bit 2 of prim type)
SQ_SX0_pc_wr_en	SQ→SX0	8	Write enable for the PC memories
SQ_SX~~x~~1_pc_wr_en	SQ→~~SXx~~SX1	~~1~~8	Write enable for the PC memories
SQ_SXx_pc_wr_addr	SQ→SXx	7	Write address for the PCs
SQ_SXx_pc_channel_mask	SQ→SXx	4	Channel mask
SQ_SXx_pc_ptr_valid	SQ→SXx	1	Read pointers are valid.
SQ_SPx_interp_valid	SQ→SPx	1	Interpolation control valid

23.2.4 SQ to SP: Staging Register Data

This is a broadcast bus that sends the VSISR information to the staging registers of the shader pipes.

Name	Direction	Bits	Description
SQ_SPx_vsr_data	SQ→SPx	96	Pointers of indexes or HOS surface information
SQ_SPx_vsr_double	SQ→SPx	1	0: Normal 96 bits per vert 1: double 192 bits per vert
SQ_SP0_ vsr_valid	SQ→SP0	1	Data is valid
SQ_SP1_ vsr_ valid	SQ→SP1	1	Data is valid
SQ_SP2_ vsr_ valid	SQ→SP2	1	Data is valid
SQ_SP3_ vsr_ valid	SQ→SP3	1	Data is valid
SQ_SPx_vsr_read	SQ→SPx	1	Increment the read pointers

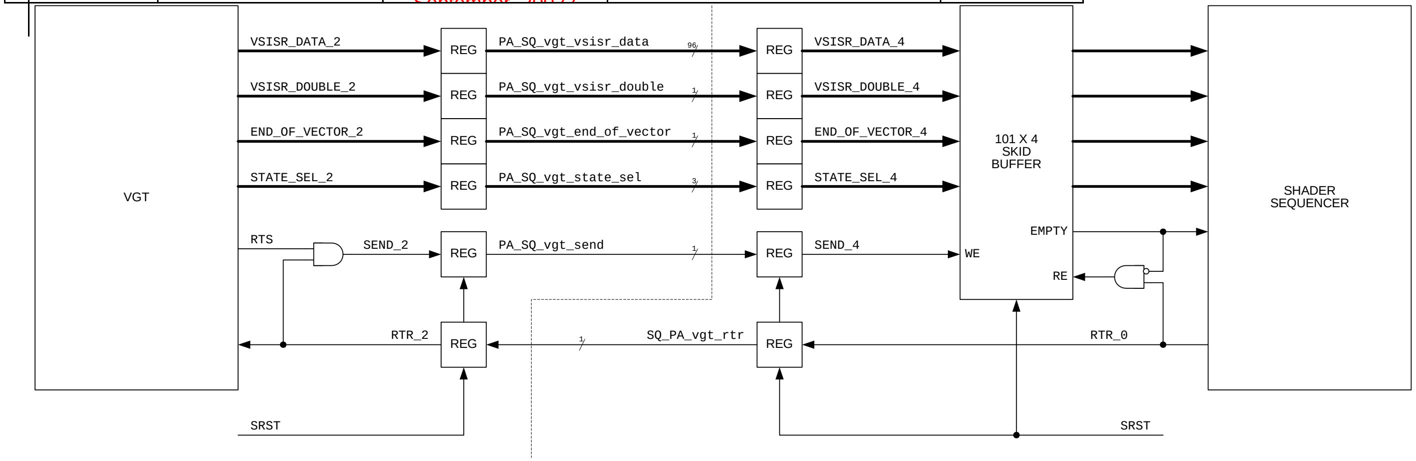
23.2.5 VGT to SQ : Vertex interface

23.2.5.1 Interface Signal Table

The area difference between the two methods is not sufficient to warrant complicating the interface or the state requirements of the VSISRs. **Therefore, the POR for this interface is that the VGT will transmit the data to the VSISRs (via the Shader Sequencer) in full, 32-bit floating-point format.** The VGT can transmit up to six 32-bit floating-point values to each VSISR where four or more values require two transmission clocks. The data bus is 96 bits wide.

Name	Bits	Description
VGT_SQ_vsisr_data	96	Pointers of indexes or HOS surface information
VGT_SQ_event	1	VGT is sending an event
VGT_SQ_vsisr_continued	1	0: Normal 96 bits per vert 1: double 192 bits per vert
VGT_SQ_end_of_vtx_vect	1	Indicates the last VSISR data set for the current process vector (for double vector data, "end_of_vector" is set on the first vector)
VGT_SQ_indx_valid	1	Vsisr data is valid
VGT_SQ_state	3	Render State (6*3+3 for constants). This signal is guaranteed to be correct when “VGT_SQ_vgt_end_of_vector” is high.
VGT_SQ_send	1	Data on the VGT_SQ is valid receive (see write-up for standard R400 SEND/RTR interface handshaking)
SQ_VGT_rtr	1	Ready to receive (see write-up for standard R400 SEND/RTR interface handshaking)

23.2.5.2 Interface Diagrams



VGT
VSISR_DATA_2
VSISR_DOUBLE_2
END_OF_VECTOR_2
STATE_SEL_2
RTS
SEND_2
RTR_2
SRST
REG
PA_SQ_vgt_vsisr_data
96
PA_SQ_vgt_vsisr_double
1
PA_SQ_vgt_end_of_vector
1
PA_SQ_vgt_state_sel
3
PA_SQ_vgt_send
1
SQ_PA_vgt_rtr
1
VSISR_DATA_4
VSISR_DOUBLE_4
END_OF_VECTOR_4
STATE_SEL_4
SEND_4
101 X 4
SKID
BUFFER
EMPTY
WE
RE
RTR_0
SRST
SHADER
SEQUENCER

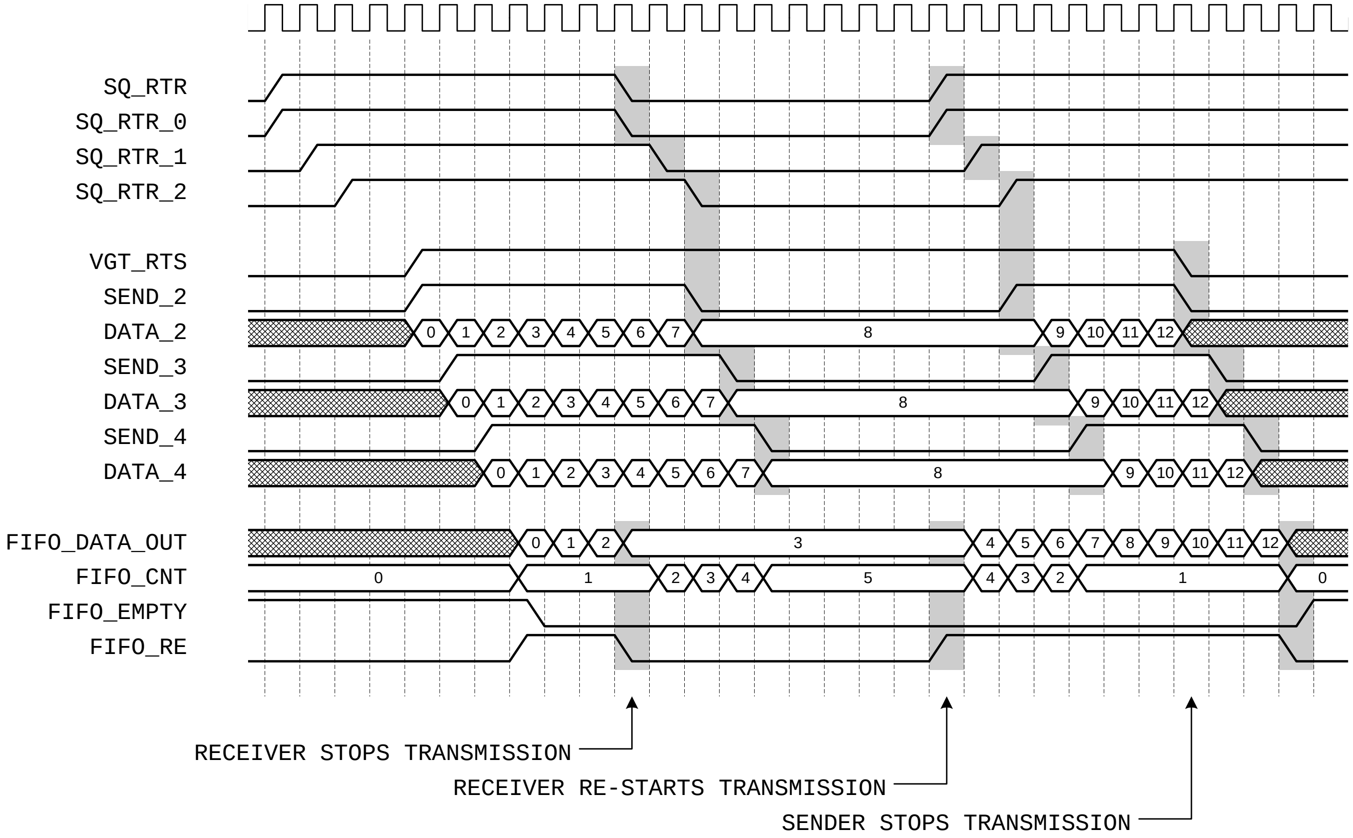


Figure 1. Detailed Logical Diagram for PA SQ vgt Interface.

23.2.6 *SQ to SX: Control bus*

Name	Direction	Bits	Description
SQ_SXx_exp_type	SQ→SXx	2	00: Pixel without z (1 to 4 buffers) 01: Pixel with z (1 to 4 buffers) 10: Position (1 or 2 results) 11: Pass thru (4,8 or 12 results aligned)
SQ_SXx_exp_number	SQ→SXx	2	Number of locations needed in the export buffer (encoding depends on the type see bellow).
SQ_SXx_exp_alu_id	SQ→SXx	1	ALU ID
SQ_SXx_exp_valid	SQ→SXx	1	Valid bit
SQ_SXx_exp_state	SQ→SXx	3	State Context
SQ_SXx_free_done	SQ→SXx	1	Pulse that indicates that the previous export is finished **from the point of view of the SP. This does not necessarily mean that the data has been transferred to RB or PA, or that the space in export buffer for that particular vector thread has been freed up.**
SQ_SXx_free_alu_id	SQ→SXx	1	ALU ID

Depending on the type the number of export location changes:

- Type 00 : Pixels without Z
  - 00 = 1 buffer
  - 01 = 2 buffers
  - 10 = 3 buffers
  - 11 = 4 buffer
- Type 01: Pixels with Z
  - 00 = 2 Buffers (color + Z)
  - 01 = 3 buffers (2 color + Z)
  - 10 = 4 buffers (3 color + Z)
  - 11 = 5 buffers (4 color + Z)
- Type 10 : Position export
  - 00 = 1 position
  - 01 = 2 positions
  - 1X = Undefined
- Type 11: Pass Thru
  - 00 = 4 buffers
  - 01 = 8 buffers
  - 10 = 12 buffers
  - 11 = Undefined

Below the thick black line is the end of transfer packet that tells the SX that a given export is finished. The report packet **will always arrive either before or at the same time than the next export to the same ALU id**.

23.2.7 *SX to SQ : Output file control*

Name	Direction	Bits	Description
SXx_SQ_exp_count_rdy	SXx→SQ	1	Raised by SX0 to indicate that the following two fields reflect the result of the most recent export
SXx_SQ_exp_pos_avail	SXx→SQ	2~~1~~	Specifies whether there is room for another position. 00 : 0 buffers ready 01 : 1 buffer ready 10 : 2 or more buffers ready
SXx_SQ_exp_buf_avail	SXx→SQ	7	Specifies the space available in the output buffers. 0: buffers are full 1: 2K-bits available (32-bits for each of the 64

			pixels in a clause) ... 64: 128K-bits available (16 128-bit entries for each of 64 pixels) 65-127: RESERVED

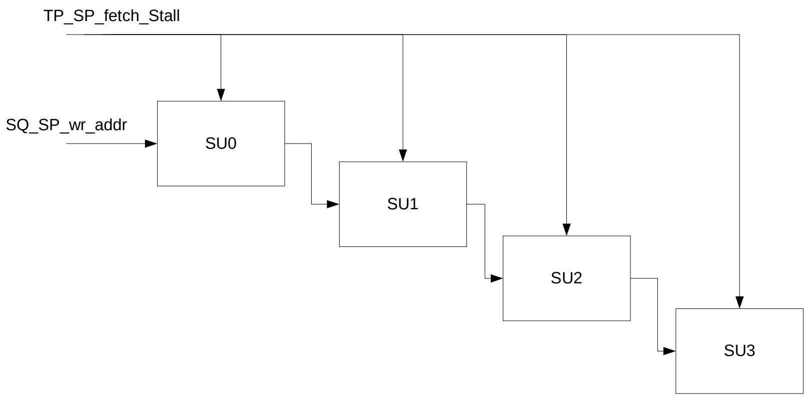
23.2.8 SQ to TP: Control bus

Once every clock, the fetch unit sends to the sequencer on which RS line it is now working and if the data in the GPRs is ready or not. This way the sequencer can update the fetch valid bits flags for the reservation station. The sequencer also provides the instruction and constants for the fetch to execute and the address in the register file where to write the fetch return data.

Name	Direction	Bits	Description
TPx_SQ_data_rdy	TPx→ SQ	1	Data ready
TPx_SQ_rs_line_num	TPx→ SQ	6	Line number in the Reservation station
TPx_SQ_type	TPx→ SQ	1	Type of data sent (0:PIXEL, 1:VERTEX)
SQ_TPx_send	SQ→TPx	1	Sending valid data
SQ_TPx_const	SQ→TPx	48	Fetch state sent over 4 clocks (192 bits total)
SQ_TPx_instr	SQ→TPx	24	Fetch instruction sent over 4 clocks
SQ_TPx_end_of_group	SQ→TPx	1	Last instruction of the group
SQ_TPx_Type	SQ→TPx	1	Type of data sent (0:PIXEL, 1:VERTEX)
SQ_TPx_gpr_phase	SQ→TPx	2	Write phase signal
SQ_TP0_lod_correct	SQ→TP0	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP0_pix_mask	SQ→TP0	4	Pixel mask 1 bit per pixel
SQ_TP1_lod_correct	SQ→TP1	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP1_pix_mask	SQ→TP1	4	Pixel mask 1 bit per pixel
SQ_TP2_lod_correct	SQ→TP2	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP2_pix_mask	SQ→TP2	4	Pixel mask 1 bit per pixel
SQ_TP3_lod_correct	SQ→TP3	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP3_pix_mask	SQ→TP3	4	Pixel mask 1 bit per pixel
SQ_TPx_rs_line_num	SQ→TPx	6	Line number in the Reservation station
SQ_TPx_write_gpr_index	SQ->TPx	7	Index into Register file for write of returned Fetch Data

23.2.9 TP to SQ: Texture stall

The TP sends this signal to the SQ and the SPs when its input buffer is full.



Name	Direction	Bits	Description
TP_SQ_fetch_stall	TP→ SQ	1	Do not send more texture request if asserted

23.2.10 *SQ to SP: Texture stall*

Name	Direction	Bits	Description
SQ_SPx_fetch_stall	SQ→SPx	1	Do not send more texture request if asserted

23.2.11 *SQ to SP: GPR and auto counter*

Name	Direction	Bits	Description
SQ_SPx_gpr_wr_addr	SQ→SPx	7	Write address
SQ_SPx_gpr_rd_addr	SQ→SPx	7	Read address
SQ_SPx_gpr_rd_en	SQ→SPx	1	Read Enable
SQ_SP0_gpr_wr_en	SQ→SPx	~~1~~4	Write Enable for the GPRs of SP0
SQ_SP1_gpr_wr_en	SQ→SPx	~~1~~4	Write Enable for the GPRs of SP1
SQ_SP2_gpr_wr_en	SQ→SPx	~~1~~4	Write Enable for the GPRs of SP2
SQ_SP3_gpr_wr_en	SQ→SPx	~~1~~4	Write Enable for the GPRs of SP3
SQ_SPx_gpr_phase	SQ→SPx	2	The phase mux (arbitrates between inputs, ALU SRC reads and writes)
SQ_SPx_channel_mask	SQ→SPx	4	The channel mask
SQ_SPx_gpr_input_sel	SQ→SPx	2	When the phase mux selects the inputs this tells from which source to read from: Interpolated data, VTX0, VTX1, autogen counter.
SQ_SPx_auto_count	SQ→SPx	~~12?~~21	Auto count generated by the SQ, common for all shader pipes

23.2.12 *SQ to SPx: Instructions*

Name	Direction	Bits	Description
SQ_SPx_instr_start	SQ→SPx	1	Instruction start
SQ_SP_instr	SQ→SPx	24~~2~~	Transferred over 4 cycles 0: ~~SRC A Select 2:0~~ ~~SRC A Argument Modifier 3:3~~ ~~SRC A swizzle 11:4~~ ~~VectorDst 17:12~~ ~~Per channel use mask (PV/Reg) 21:18~~SRC A Negate Argument Modifier 0:0 SRC A Abs Argument Modifier 1:1 SRC A Swizzle 9:2 Vector Dst 15:10 Per channel Select 23:16 00: GPR 01: PV 10: PS 11: Constant (if 11 has to be 11 for all channels) -- - 1: SRC B Negate Argument Modifier 0:0 SRC B Abs Argument Modifier 1:1 SRC B Swizzle 9:2 Scalar Dst 15:10 Per channel Select 23:16 00: GPR 01: PV 10: PS 11: Constant (if 11 has to be 11 for all channels) ~~SRC B Select 2:0~~ ~~SRC B Argument Modifier 3:3~~ ~~SRC B swizzle 11:4~~ ~~ScalarDst 17:12~~ ~~Per channel use mask (PV/Reg) 21:18~~ -- - 2: SRC C Negate Argument Modifier 0:0 SRC C Abs Argument Modifier 1:1 SRC C Swizzle 9:2 Unused 15:10 Per channel Select 23:16 00: GPR 01: PV 10: PS 11: Constant (if 11 has to be 11 for all channels) ~~SRC C Select 2:0~~ ~~SRC C Argument Modifier 3:3~~ ~~SRC C swizzle 11:4~~ ~~Per channel use mask (PV/Reg) 21:18~~ -- - 3: Vector Opcode 4:0 Scalar Opcode 10:5 Vector Clamp 11:11 Scalar Clamp 12:12 Vector Write Mask 16:13 Scalar Write Mask 20:17 Unused 23:21

SQ_SP0_pred_override	SQ→SP0	4	0: Use per channel RGBA field (enables the per channel logic, if not set only pay attention to the 11 seting). 1: Use GPR
SQ_SP1_pred_override	SQ→SP1	4	0: Use per channel RGBA field (enables the per channel logic, if not set only pay attention to the 11 seting). 1: Use GPR
SQ_SP2_pred_override	SQ→SP2	4	0: Use per channel RGBA field (enables the per channel logic, if not set only pay attention to the 11 seting). 1: Use GPR
SQ_SP3_pred_override	SQ→SP3	4	0: Use per channel RGBA field (enables the per channel logic, if not set only pay attention to the 11 seting). 1: Use GPR
SQ_SPx_exp_~~alu~~_id	SQ→SPx	1	GPR~~ALU~~ ID
SQ_SPx_exporting	SQ→SPx	1	0: Not Exporting 1: Exporting
SQ_SPx_stall	SQ→SPx	1	Stall signal

23.2.13 *SQ to SX: write mask interface (must be aligned with the SP data)*

Formatted: Bullets and Numbering

Name	**Direction**	**Bits**	**Description**
SQ_SX0_write_mask	SQ→SP0	8	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock. This is for the data coming of SP0 and SP2.
SQ_SX1 write_mask	SQ→SP1	8	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock. This is for the data coming of SP1 and SP3.

~~23.2.13~~23.2.14 *SP to SQ: Constant address load/ Predicate Set/Kill set*

Formatted: Bullets and Numbering

Name	**Direction**	**Bits**	**Description**
SP0_SQ_const_addr	SP0→SQ	36	Constant address load / predicate vector load (4 bits only)/ Kill vector load (4 bits only) to the sequencer
SP0_SQ_valid	SP0→SQ	1	Data valid
SP1_SQ_const_addr	SP1→SQ	36	Constant address load / predicate vector load (4 bits only)/ Kill vector load (4 bits only) to the sequencer
SP1_SQ_valid	SP1→SQ	1	Data valid
SP2_SQ_const_addr	SP2→SQ	36	Constant address load / predicate vector load (4 bits only)/ Kill vector load (4 bits only) to the sequencer
SP2_SQ_valid	SP2→SQ	1	Data valid
SP3_SQ_const_addr	SP3→SQ	36	Constant address load / predicate vector load (4 bits only)/ Kill vector load (4 bits only) to the sequencer
SP3_SQ_valid	SP3→SQ	1	Data valid
SP0_SQ_data_type	SP→SQ	~~1~~2	Data Type 0: Constant Load 1: Predicate Set 2: Kill vector load

Because of the sharing of the bus none of the MOVA, PREDSET or KILL instructions may be coissued.

Formatted

~~23.2.14~~23.2.15 *SQ to SPx: constant broadcast*

Formatted: Bullets and Numbering

Name	**Direction**	**Bits**	**Description**
SQ_SPx_const	SQ→SPx	128	Constant broadcast

Formatted: Bullets and Numbering

~~23.2.15 SP0 to SQ: Kill vector load~~

23.2.16 SQ to CP: RBBM bus

Name	Direction	Bits	Description
SQ_RBB_rs	SQ→CP	1	Read Strobe
SQ_RBB_rd	SQ→CP	32	Read Data
SQ_RBBM_nrtrtr	SQ→CP	1	Optional
SQ_RBBM_rtr	SQ→CP	1	Real-Time (Optional)

23.2.17 CP to SQ: RBBM bus

Name	Direction	Bits	Description
rbbm_we	CP→SQ	1	Write Enable
rbbm_a	CP→SQ	15	Address -- Upper Extent is TBD (16:2)
rbbm_wd	CP→SQ	32	Data
rbbm_be	CP→SQ	4	Byte Enables
rbbm_re	CP→SQ	1	Read Enable
rbb_rs0	CP→SQ	1	Read Return Strobe 0
rbb_rs1	CP→SQ	1	Read Return Strobe 1
rbb_rd0	CP→SQ	32	Read Data 0
rbb_rd1	CP→SQ	32	Read Data 0
RBBM_SQ_soft_reset	CP→SQ	1	Soft Reset

23.2.18 SQ to CP: State report

Name	Direction	Bits	Description
SQ_CP_vs_event	SQ→CP	1	Vertex Shader Event
SQ_CP_vs_eventid	SQ→CP	4~~2~~	Vertex Shader Event ID
SQ_CP_ps_event	SQ→CP	1	Pixel Shader Event
SQ_CP_ps_eventid	SQ→CP	4~~2~~	Pixel Shader Event ID

~~eventid = 0 => *sEndOfState (i.e. VsEndOfState)~~
~~eventid = 1 => *sDone (i.e. VsDone)~~

~~So, the CP will assume the Vs is done with a state whenever it gets a pulse on the SQ_CP_vs_event and the SQ_CP_vs_eventid = 0.~~

23.3 Example of control flow program execution

We now provide some examples of execution to better illustrate the new design.

Given the program:

Alu 0
Alu 1
Tex 0
Tex 1
Alu 3 Serial
Alu 4
Tex 2
Alu 5
Alu 6 Serial
Tex 3
Alu 7
Alloc Position 1 buffer
Alu 8 Export
Tex 4

Alloc Parameter 3 buffers
Alu 9 Export 0
Tex 5
Alu 10 Serial Export 2
Alu 11 Export 1 End

Would be converted into the following CF instructions:

```
Execute 0 Alu 0 Alu 0 Tex 0 Tex 1 Alu 0 Alu 0 Tex 0 Alu 1 Alu 0 Tex
Execute 0 Alu
Alloc Position 1
Execute 0 Alu 0 Tex
Alloc Param 3
Execute_end 0 Alu 0 Tex 1 Alu 0 Alu
```

And the execution of this program would look like this:

Put thread in Vertex RS:

Control Flow Instruction Pointer (12 bits), (CFP)
Execution Count Marker (3 or 4 bits), (ECM)
Loop Iterators (4x9 bits), (LI)
Call return pointers (4x12 bits), (CRP)
Predicate Bits(4x64 bits), (PB)
Export ID (1 bit), (EXID)
GPR Base Ptr (8 bits), (GPR)
Export Base Ptr (7 bits), (EB)
Context Ptr (3 bits).(CPTR)
LOD correction bits (16x6 bits) (LOD)

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	0	0	0	0	0	0	0	0	0

Valid Thread (VALID)
Texture/ALU engine needed (TYPE)
Texture Reads are outstanding (PENDING)
Waiting on Texture Read to Complete (SERIAL)
Allocation Wait (2 bits) (ALLOC)
  00 – No allocation needed
  01 – Position export allocation needed (ordered export)
  10 – Parameter or pixel export needed (ordered export)
  11 – pass thru (out of order export)
Allocation Size (4 bits) (SIZE)
Position Allocated (POS_ALLOC)
First thread of a new context (FIRST)
Last (1 bit), (LAST)

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	0	0	0	0	0	1	0

Then the thread is picked up for the execution of the first control flow instruction:

```
Execute 0 Alu 0 Alu 0 Tex 0 Tex 1 Alu 0 Alu 0 Tex 0 Alu 1 Alu 0 Tex
```

It executes the first two ALU instructions and goes back to the RS for a resource request change. Here is the state returned to the RS:

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	2	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	0	0	0	0	0	1	0

Then when the texture pipe frees up, the arbiter picks up the thread to issue the texture reads. The thread comes back in this state:

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	4	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	1	0	0	0	1	0

Because of the serial bit the arbiter must wait for the texture to return and clear the PENDING bit before it can pick the thread up. Lets say that the texture reads are complete, then the arbiter picks up the thread and returns it in this state:

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	6	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	0	0	0	0	0	1	0

Again the TP frees up, the arbiter picks up the thread and executes. It returns in this state:

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	7	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	0	0	0	1	0

Now, even if the texture has not returned we can still pick up the thread for ALU execution because the serial bit is not set. The thread will however come back to the RS for the second ALU instruction because it has the serial bit set.

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	8	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	1	0	0	0	1	0

As soon as the TP clears the pending bit the thread is picked up and returns:

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	9	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	0	0	0	0	0	1	0

Picked up by the TP and returns:

```
Execute 0 Alu
```

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
1	0	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	0	0	0	1	0

Picked up by the ALU and returns (lets say the TP has not returned yet):

```
Alloc Position 1
```

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
2	0	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	01	1	0	1	0

If the SX has the place for the export, the SQ is going to allocate and pick up the thread for execution. It returns to the RS in this state:

```
Execute 0 Alu 0 Tex
```

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
3	1	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	1	0	0	0	1	1	0

Now, since the TP has not returned yet, we must wait for it to return because we cannot issue multiple texture requests. The TP returns, clears the PENDING bit and we proceed:

```
Alloc Param 3
```

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
4	0	0	0	0	1	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	10	3	1	1	0

Once again the SQ makes sure the SX has enough room in the Parameter cache before it can pick up this thread.

```
Execute_end 0 Alu 0 Tex 1 Alu 0 Alu
```

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
5	1	0	0	0	1	0	100	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	1	0	0	0	1	1	0

This executes on the TP and then returns:

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
5	2	0	0	0	1	0	100	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	1	0	0	1	1	1

Waits for the TP to return because of the textures reads are pending (and SERIAL in this case). Then executes and does not return to the RS because the LAST bit is set. This is the end of this thread and before dropping it on the floor, the SQ notifies the SX of export completion.

24. Open issues

Need to do some testing on the size of the register file as well as on the register file allocation method (dynamic VS static).

Saving power?

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 2015~~11~~ October, 200210	GEN-CXXXXX-REVA	1 of 51

Author: Laurent Lefebvre

Issue To:	**Copy No:**

R400 Sequencer Specification

SQ

Version 2.06~~5~~

Overview: This is an architectural specification for the R400 Sequencer block (SEQ). It provides an overview of the required capabilities and expected uses of the block. It also describes the block interfaces, internal sub-blocks, and provides internal state diagrams.

AUTOMATICALLY UPDATED FIELDS:
Document Location: C:\perforce\r400\doc_lib\design\blocks\sq\R400_Sequencer.doc
Current Intranet Search Title: R400 Sequencer Specification

APPROVALS	
Name/Dept	Signature/Date

Remarks:

THIS DOCUMENT CONTAINS CONFIDENTIAL INFORMATION THAT COULD BE SUBSTANTIALLY DETRIMENTAL TO THE INTEREST OF ATI TECHNOLOGIES INC. THROUGH UNAUTHORIZED USE OR DISCLOSURE.

ATI 2034
LG v. ATI
IPR2015-00326

ATI Ex. 2119
IPR2023-00922

Table Of Contents

1. OVERVIEW 7
1.1 Top Level Block Diagram 9
1.2 Data Flow graph (SP) 10
1.3 Control Graph 11
2. INTERPOLATED DATA BUS 11
3. INSTRUCTION STORE 14
4. SEQUENCER INSTRUCTIONS 14
5. CONSTANT STORES 14
5.1 Memory organizations 14
5.2 Management of the Control Flow Constants 15
5.3 Management of the re-mapping tables 15
5.3.1 R400 Constant management 15
5.3.2 Proposal for R400LE constant management 15
5.3.3 Dirty bits 17
5.3.4 Free List Block 17
5.3.5 De-allocate Block 18
5.3.6 Operation of Incremental model 18
5.4 Constant Store Indexing 18
5.5 Real Time Commands 19
5.6 Constant Waterfalling 19
6. LOOPING AND BRANCHES 20
6.1 The controlling state. 20
6.2 The Control Flow Program 20
6.2.1 Control flow instructions table 21
6.3 Implementation 23
6.4 Data dependant predicate instructions 24
6.5 HW Detection of PV,PS 25
6.6 Register file indexing 25
6.7 Debugging the Shaders ..... 26~~25~~
6.7.1 Method 1: Debugging registers 26
6.7.2 Method 2: Exporting the values in the GPRs 26
7. PIXEL KILL MASK 26
8. MULTIPASS VERTEX SHADERS (HOS) 26
9. REGISTER FILE ALLOCATION ..... 27~~26~~
10. FETCH ARBITRATION 28
11. ALU ARBITRATION 28
12. HANDLING STALLS 29
13. CONTENT OF THE RESERVATION STATION FIFOS 29
14. THE OUTPUT FILE 29
15. IJ FORMAT 29
15.1 Interpolation of constant attributes 29
16. STAGING REGISTERS 30

17. THE PARAMETER CACHE 31
17.1 Export restrictions 32
17.1.1 Pixel exports: 32
17.1.2 Vertex exports: 32
17.1.3 Pass thru exports: 32
17.2 Arbitration restrictions 32
18. EXPORT TYPES 32
18.1 Vertex Shading 32
18.2 Pixel Shading 33
19. SPECIAL INTERPOLATION MODES 33
19.1 Real time commands 33
19.2 Sprites/ XY screen coordinates/ FB information 33
19.3 Auto generated counters 34
19.3.1 Vertex shaders 34
19.3.2 Pixel shaders 34
20. STATE MANAGEMENT 35
20.1 Parameter cache synchronization 35
21. XY ADDRESS IMPORTS 35
21.1 Vertex indexes imports 35
22. REGISTERS 35
23. INTERFACES 36
23.1 External Interfaces 36
23.2 SC to SP Interfaces 36
23.2.1 SC_SP# 36
23.2.2 SC_SQ 37
23.2.3 SQ to SX(SP): Interpolator bus 39
23.2.4 SQ to SP: Staging Register Data 39
23.2.5 VGT to SQ : Vertex interface 39
23.2.6 SQ to SX: Control bus 42
23.2.7 SX to SQ : Output file control 42
23.2.8 SQ to TP: Control bus 43
23.2.9 TP to SQ: Texture stall 43
23.2.10 SQ to SP: Texture stall 44
23.2.11 SQ to SP: GPR and auto counter 44
23.2.12 SQ to SPx: Instructions 45
23.2.13 SP to SQ: Constant address load/ Predicate Set/Kill set 46
23.2.14 SQ to SPx: constant broadcast 46
23.2.15 SQ to CP: RBBM bus 46
23.2.16 CP to SQ: RBBM bus 46
23.2.17 SQ to CP: State report 47
23.3 Example of control flow program execution 47

24. OPEN ISSUES.. 51

Revision Changes:

Rev 0.1 (Laurent Lefebvre) Date: May 7, 2001	First draft.
Rev 0.2 (Laurent Lefebvre) Date : July 9, 2001	Changed the interfaces to reflect the changes in the SP. Added some details in the arbitration section.
Rev 0.3 (Laurent Lefebvre) Date : August 6, 2001	Reviewed the Sequencer spec after the meeting on August 3, 2001.
Rev 0.4 (Laurent Lefebvre) Date : August 24, 2001	Added the dynamic allocation method for register file and an example (written in part by Vic) of the flow of pixels/vertices in the sequencer.
Rev 0.5 (Laurent Lefebvre) Date : September 7, 2001	Added timing diagrams (Vic)
Rev 0.6 (Laurent Lefebvre) Date : September 24, 2001	Changed the spec to reflect the new R400 architecture. Added interfaces.
Rev 0.7 (Laurent Lefebvre) Date : October 5, 2001	Added constant store management, instruction store management, control flow management and data dependant predication.
Rev 0.8 (Laurent Lefebvre) Date : October 8, 2001	Changed the control flow method to be more flexible. Also updated the external interfaces.
Rev 0.9 (Laurent Lefebvre) Date : October 17, 2001	Incorporated changes made in the 10/18/01 control flow meeting. Added a NOP instruction, removed the conditional_execute_or_jump. Added debug registers.
Rev 1.0 (Laurent Lefebvre) Date : October 19, 2001	Refined interfaces to RB. Added state registers.
Rev 1.1 (Laurent Lefebvre) Date : October 26, 2001	Added SEQ→SP0 interfaces. Changed delta precision. Changed VGT→SP0 interface. Debug Methods added.
Rev 1.2 (Laurent Lefebvre) Date : November 16, 2001	Interfaces greatly refined. Cleaned up the spec.
Rev 1.3 (Laurent Lefebvre) Date : November 26, 2001	Added the different interpolation modes.
Rev 1.4 (Laurent Lefebvre) Date : December 6, 2001	Added the auto incrementing counters. Changed the VGT→SQ interface. Added content on constant management. Updated GPRs.
Rev 1.5 (Laurent Lefebvre) Date : December 11, 2001	Removed from the spec all interfaces that weren't directly tied to the SQ. Added explanations on constant management. Added PA→SQ synchronization fields and explanation.
Rev 1.6 (Laurent Lefebvre) Date : January 7, 2002	Added more details on the staging register. Added detail about the parameter caches. Changed the call instruction to a Conditionnal_call instruction. Added details on constant management and updated the diagram.
Rev 1.7 (Laurent Lefebvre) Date : February 4, 2002	Added Real Time parameter control in the SX interface. Updated the control flow section.
Rev 1.8 (Laurent Lefebvre) Date : March 4, 2002	New interfaces to the SX block. Added the end of clause modifier, removed the end of clause instructions.
Rev 1.9 (Laurent Lefebvre) Date : March 18, 2002	Rearangement of the CF instruction bits in order to ensure byte alignement.
Rev 1.10 (Laurent Lefebvre) Date : March 25, 2002	Updated the interfaces and added a section on exporting rules.
Rev 1.11 (Laurent Lefebvre) Date : April 19, 2002	Added CP state report interface. Last version of the spec with the old control flow scheme
Rev 2.0 (Laurent Lefebvre) Date : April 19, 2002	New control flow scheme

Rev 2.01 (Laurent Lefebvre) Date : May 2, 2002	Changed slightly the control flow instructions to allow force jumps and calls.
Rev 2.02 (Laurent Lefebvre) Date : May 13, 2002	Updated the Opcodes. Added type field to the constant/pred interface. Added Last field to the SQ→SP instruction load interface.
Rev 2.03 (Laurent Lefebvre) Date : July 15, 2002	SP interface updated to include predication optimizations. Added the predicate no stall instructions,
Rev 2.04 (Laurent Lefebvre) Date :August 2, 2002	Documented the new parameter generation scheme for XY coordinates points and lines STs.
Rev 2.05 (Laurent Lefebvre) Date : September 10, 2002	Some interface changes and an architectural change to the auto-counter scheme.
Rev 2.06 (Laurent Lefebvre) Date : October 11, 2002	Widened the event interface to 5 bits. Some other little typos corrected.

1. Overview

The sequencer chooses two ALU threads and a fetch hread to execute, and executes all of the instructions in a block before looking for a new clause of the same type. Two ALU threads are executed interleaved to hide the ALU latency. The arbitrator will give priority to older threads. There are two separate reservation stations, one for pixel vectors and one for vertices vectors. This way a pixel can pass a vertex and a vertex can pass a pixel.

To support the shader pipe the sequencer also contains the shader instruction cache, constant store, control flow constants and texture state. The four shader pipes also execute the same instruction thus there is only one sequencer for the whole chip.

The sequencer first arbitrates between vectors of 64 vertices that arrive directly from primitive assembly and vectors of 16 quads (64 pixels) that are generated in the scan converter.

The vertex or pixel program specifies how many GPRs it needs to execute. The sequencer will not start the next vector until the needed space is available in the GPRs.

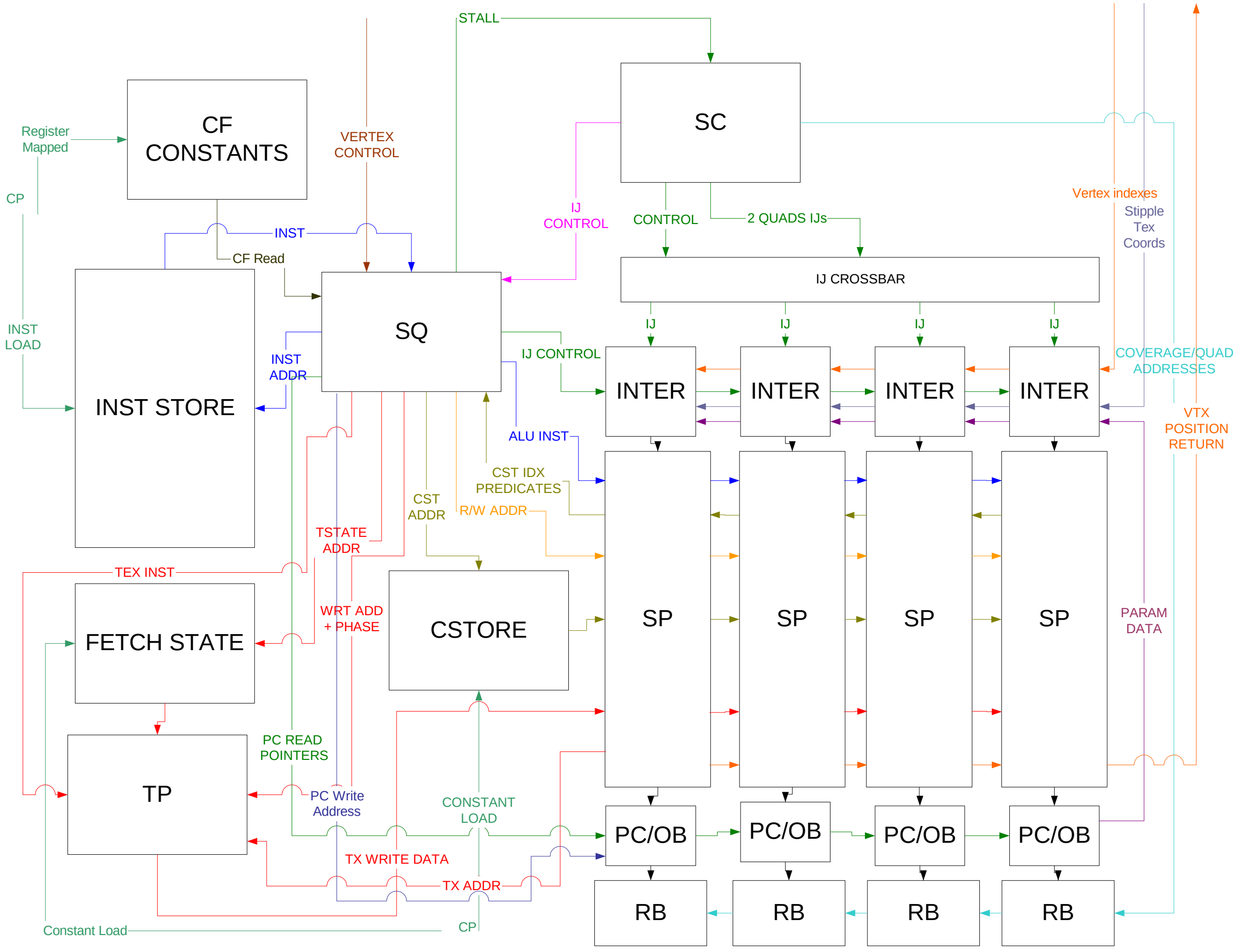


Figure 1: General Sequencer overview

1.1 Top Level Block Diagram

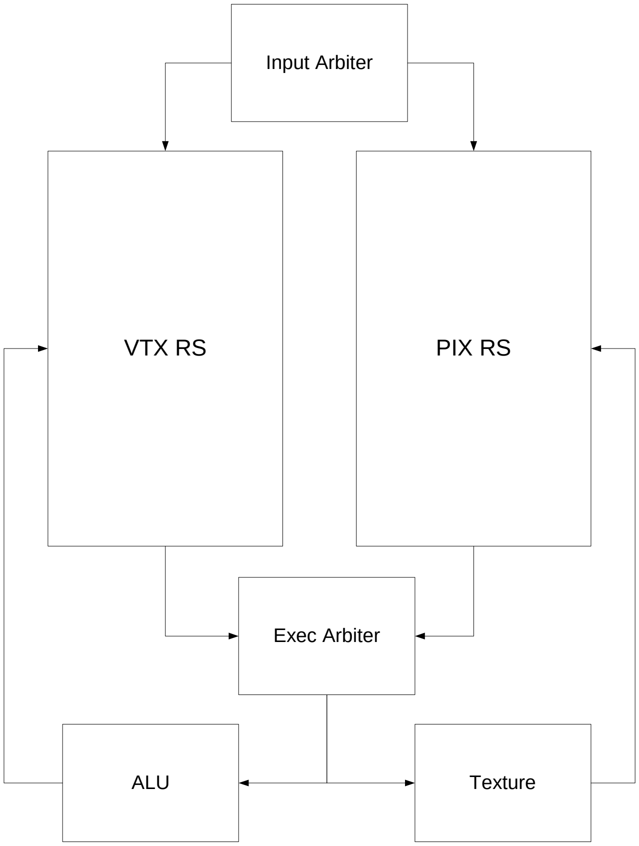


Figure 2: Reservation stations and arbiters

Under this new scheme, the sequencer (SQ) will only use one global state management machine per vector type (pixel, vertex) that we call the reservation station (RS).

1.2 Data Flow graph (SP)

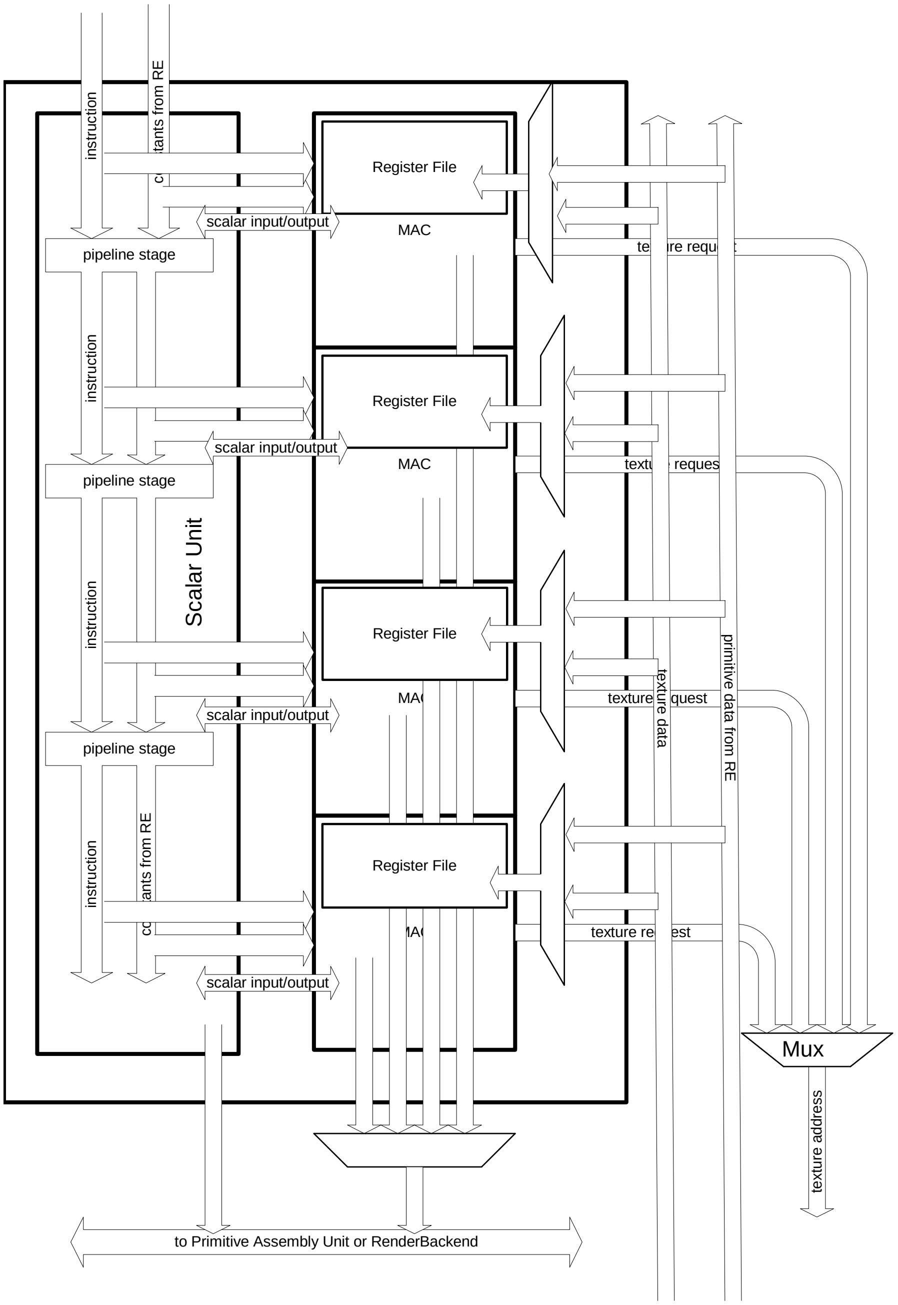


Figure 3: The shader Pipe

The gray area represents blocks that are replicated 4 times per shader pipe (16 times on the overall chip).

1.3 Control Graph

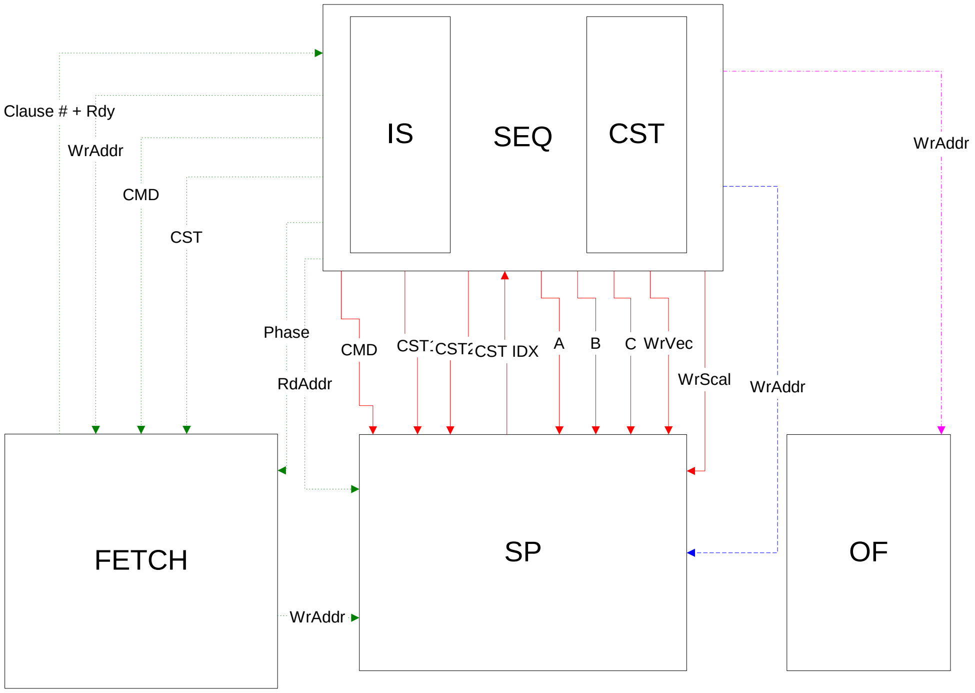
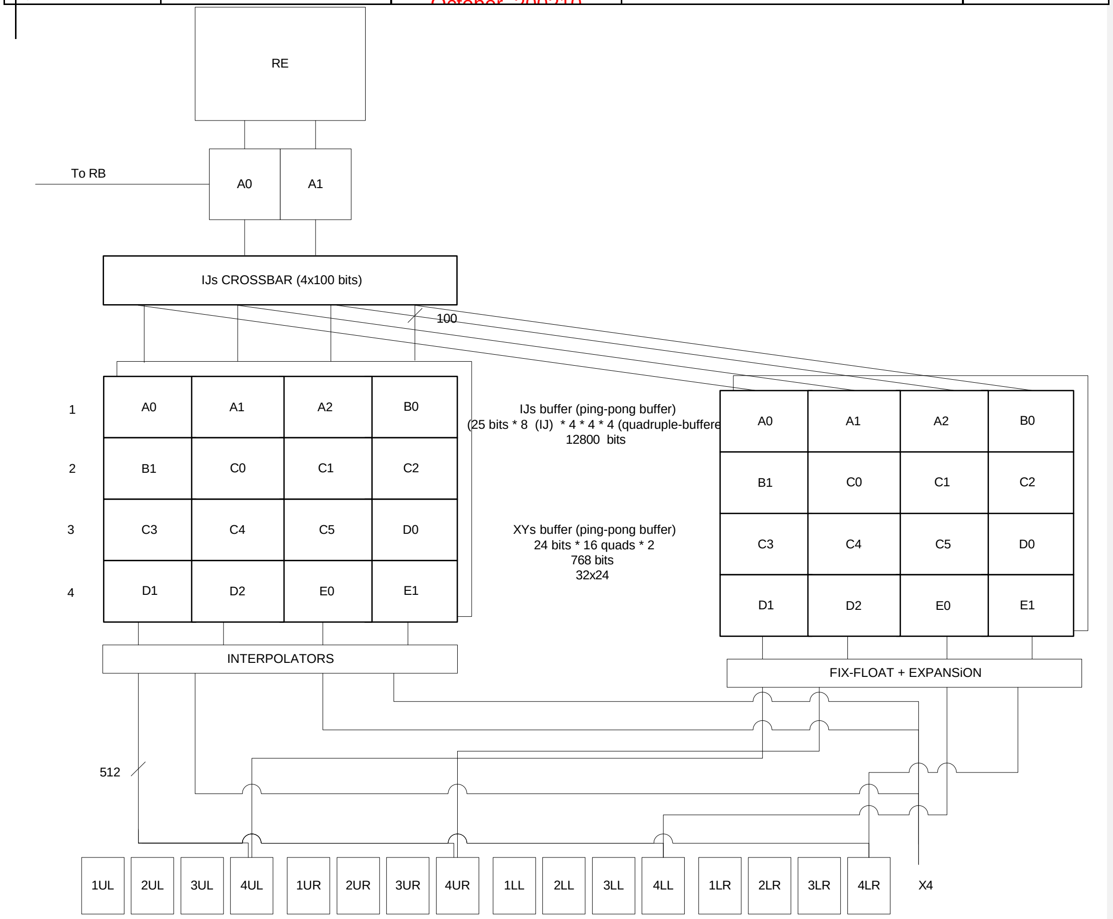


Figure 4: Sequencer Control interfaces

In green is represented the Fetch control interface, in red the ALU control interface, in blue the Interpolated/Vector control interface and in purple is the output file control interface.

2. Interpolated data bus

The interpolators contain an IJ buffer to pack the information as much as possible before writing it to the register file.



1	A0	A1	A2	B0
2	B1	C0	C1	C2
3	C3	C4	C5	D0
4	D1	D2	E0	E1

A0	A1	A2	B0
B1	C0	C1	C2
C3	C4	C5	D0
D1	D2	E0	E1

Figure 5: Interpolation buffers

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21	T22	T23
											WRITES													
SP 0	A0	A0	XY A0	B1	B1	XY B1	C3	C3	XY C3				D1	D1	XY D1									
SP 1	A1	A1	XY A1				C0	C0	XY C0	C4	C4	XY C4	D2	D2	XY D2									
SP 2	A2	A2	XY A2				C1	C1	XY C1	C5	C5	XY C5				E0	E0	XY E0						
SP 3				B0	B0	XY B0	C2	C2	XY C2				D0	D0	XY D0	E1	E1	XY E1						
											READS													
SP 0	XY 0-3	XY 16-19	XY 32-35	XY 48-51	A0	B1	C3	D1					A0	B1	C3	D1					V 0-3	V 16-19	V 32-35	V 48-51
SP 1	XY 4-7	XY 20-23	XY 36-39	XY 52-55	A1		C4	D2		C0			A1		C4	D2		C0			V 4-7	V 20-23	V 36-39	V 52-55
SP 2	XY 8-11	XY 24-27	XY 40-43	XY 56-59	A2		C5			C1		E0	A2		C5			C1		E0	V 8-11	V 24-27	V 40-43	V 56-59
SP 3	XY 12-15	XY 28-31	XY 44-47	XY 60-63					B0	C2	D0	E1					B0	C2	D0	E1	V 12-15	V 28-31	V 44-47	V 60-63
	XY				P1								P2								VTX			

Figure 6: Interpolation timing diagram

Above is an example of a tile the sequencer might receive from the SC. The write side is how the data get stacked into the XY and IJ buffers, the read side is how the data is passed to the GPRs. The IJ information is packed in the IJ buffer 4 quads at a time or two clocks. The sequencer allows at any given time as many as four quads to interpolate a parameter. They all have to come from the same primitive. Then the sequencer controls the write mask to the GPRs to write the valid data in.

3. Instruction Store

There is going to be only one instruction store for the whole chip. It will contain 4096 instructions of 96 bits each.

It is likely to be a 1 port memory; we use 1 clock to load the ALU instruction, 1 clocks to load the Fetch instruction, 1 clock to load 2 control flow instructions and 1 clock to write instructions.

The instruction store is loaded by the CP thru the register mapped registers.

The VS_BASE and PS_BASE context registers are used to specify for each context where its shader is in the instruction memory.

For the Real time commands the story is quite the same but for some small differences. There are no wrap-around points for real time so the driver must be careful not to overwrite regular shader data. The shared code (shared subroutines) uses the same path as real time.

4. Sequencer Instructions

All control flow instructions and move instructions are handled by the sequencer only. The ALUs will perform NOPs during this time (MOV PV,PV, PS,PS) if they have nothing else to do.

5. Constant Stores

5.1 Memory organizations

A likely size for the ALU constant store is 1024x128 bits. The read BW from the ALU constant store is 128 bits/clock and the write bandwidth is 32 bits/clock (directed by the CP bus size not by memory ports).

The maximum logical size of the constant store for a given shader is 256 constants. Or 512 for the pixel/vertex shader pair. The size of the re-mapping table is 128 lines (each line addresses 4 constants). The write granularity is 4 constants or 512 bits. It takes 16 clocks to write the four constants. Real time requires 256 lines in the physical memory (this is physically register mapped).

The texture state is also kept in a similar memory. The size of this memory is 320x96 bits (128 texture states for regular mode, 32 states for RT). The memory thus holds 128 texture states (192 bits per state). The logical size exposes 32 different states total, which are going to be shared between the pixel and the vertex shader. The size of the re-mapping table to for the texture state memory is 32 lines (each line addresses 1 texture state lines in the real memory). The CP write granularity is 1 texture state lines (or 192 bits). The driver sends 512 bits but the CP ignores the top 320 bits. It thus takes 6 clocks to write the texture state. Real time requires 32 lines in the physical memory (this is physically register mapped).

The control flow constant memory doesn't sit behind a renaming table. It is register mapped and thus the driver must reload its content each time there is a change in the control flow constants. Its size is 320*32 because it must hold 8 copies of the 32 dwords of control flow constants and the loop construct constants must be aligned.

The constant re-mapping tables for texture state and ALU constants are logically register mapped for regular mode and physically register mapped for RT operation.

5.2 Management of the Control Flow Constants

The control flow constants are register mapped, thus the CP writes to the according register to set the constant, the SQ decodes the address and writes to the block pointed by its current base pointer (CF_WR_BASE). On the read side, one level of indirection is used. A register (SQ_CONTEXT_MISC.CF_RD_BASE) keeps the current base pointer to the control flow block. This register is copied whenever there is a state change. Should the CP write to CF after the state change, the base register is updated with the (current pointer number +1)% number of states. This way, if the CP doesn't write to CF the state is going to use the previous CF constants.

5.3 Management of the re-mapping tables

5.3.1 *R400 Constant management*

The sequencer is responsible to manage two re-mapping tables (one for the constant store and one for the texture state). On a state change (by the driver), the sequencer will broadside copy the contents of its re-mapping tables to a new one. We have 8 different re-mapping tables we can use concurrently.

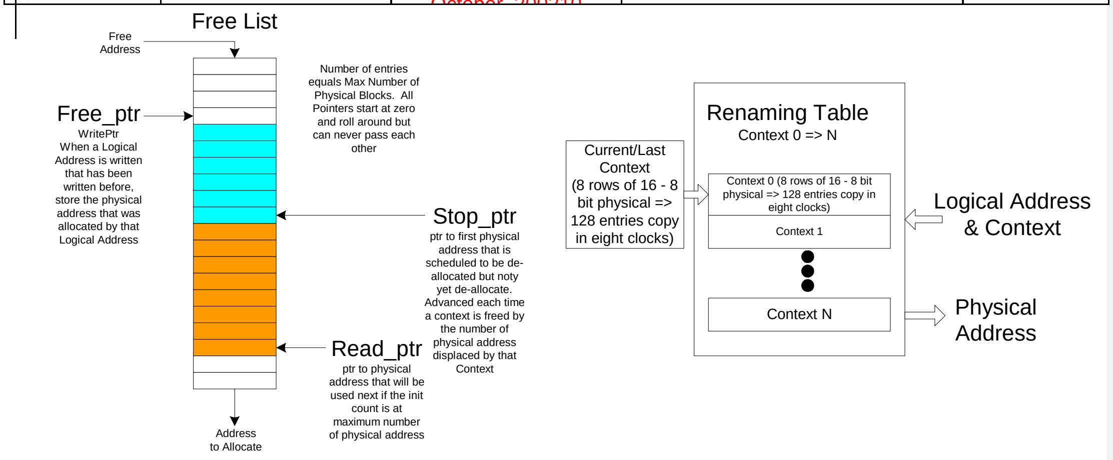
The constant memory update will be incremental, the driver only need to update the constants that actually changed between the two state changes.

For this model to work in its simplest form, the requirement is that the physical memory MUST be at least twice as large as the logical address space + the space allocated for Real Time. In our case, since the logical address space is 512 and the reserved RT space can be up to 256 entries, the memory must be of sizes 1280 and above. Similarly the size of the texture store must be of 32*2+32 = 96 entries and above.

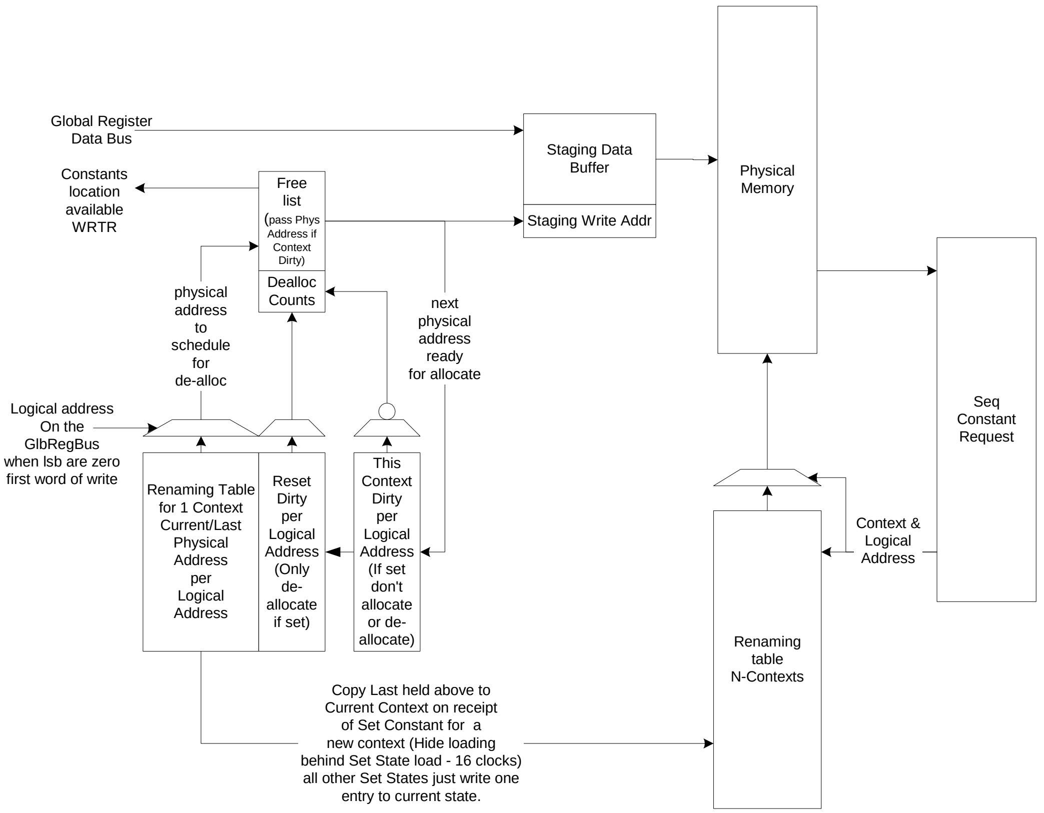
5.3.2 *Proposal for R400LE constant management*

To make this scheme work with only 512+256 = 768 entries, upon reception of a CONTROL packet of state + 1, the sequencer would check for SQ_IDLE and PA_IDLE and if both are idle will erase the content of state to replace it with the new state (this is depicted in Figure 8: De-allocation mechanism~~Figure 8: De-allocation mechanismFigure 8: De-allocation mechanism~~). Note that in the case a state is cleared a value of 0 is written to the corresponding de-allocation counter location so that when the SQ is going to report a state change, nothing will be de-allocated upon the first report.

The second path sets all context dirty bits that were used in the current state to 1 (thus allowing the new state to reuse these physical addresses if needed).



Free List
Free Address
Free_ptr
WritePtr
When a Logical Address is written that has been written before, store the physical address that was allocated by that Logical Address
Number of entries equals Max Number of Physical Blocks. All Pointers start at zero and roll around but can never pass each other
Stop_ptr
ptr to first physical address that is scheduled to be de-allocated but noty yet de-allocate. Advanced each time a context is freed by the number of physical address displaced by that Context
Read_ptr
ptr to physical address that will be used next if the init count is at maximum number of physical address
Address to Allocate
Current/Last Context (8 rows of 16 - 8 bit physical => 128 entries copy in eight clocks)
Renaming Table
Context 0 => N
Context 0 (8 rows of 16 - 8 bit physical => 128 entries copy in eight clocks)
Context 1
Context N
Logical Address & Context
Physical Address



Global Register Data Bus
Staging Data Buffer
Staging Write Addr
Physical Memory
Constants location available WRTR
Free list (pass Phys Address if Context Dirty)
Dealloc Counts
physical address to schedule for de-alloc
next physical address ready for allocate
Logical address On the GlbRegBus when lsb are zero first word of write
Renaming Table for 1 Context Current/Last Physical Address per Logical Address
Reset Dirty per Logical Address (Only de-allocate if set)
This Context Dirty per Logical Address (If set don't allocate or de-allocate)
Seq Constant Request
Context & Logical Address
Renaming table N-Contexts
Copy Last held above to Current Context on receipt of Set Constant for a new context (Hide loading behind Set State load - 16 clocks) all other Set States just write one entry to current state.

Figure 7: Constant management

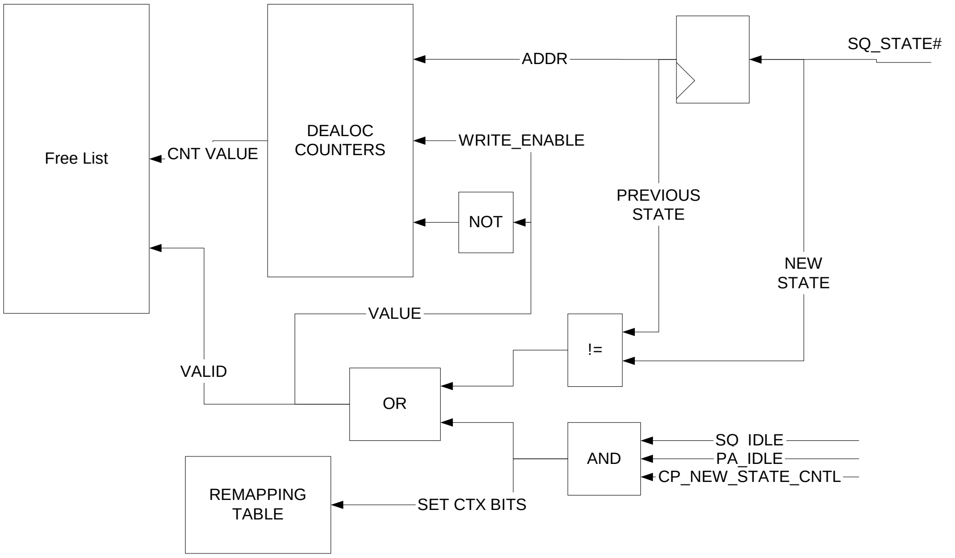


Figure 8: De-allocation mechanism for R400LE

5.3.3 *Dirty bits*

Two sets of dirty bits will be maintained per logical address. The first one will be set to zero on reset and set when the logical address is addressed. The second one will be set to zero whenever a new context is written and set for each address written while in this context. The reset dirty is not set, then writing to that logical address will not require de-allocation of whatever address stored in the renaming table. If it is set and the context dirty is not set, then the physical address store needs to be de-allocated and a new physical address is necessary to store the incoming data. If they are both set, then the data will be written into the physical address held in the renaming for the current logical address. No de-allocation or allocation takes place. This will happen when the driver does a set constant twice to the same logical address between context changes. NOTE: It is important to detect and prevent this, failure to do it will allow multiple writes to allocate all physical memory and thus hang because a context will not fit for rendering to start and thus free up space.

5.3.4 *Free List Block*

A free list block that would consist of a counter (called the IFC or Initial Free Counter) that would reset to zero and incremented every time a chunk of physical memory is used until they have all been used once. This counter would be checked each time a physical block is needed, and if the original ones have not been used up, us a new one, else check the free list for an available physical block address. The count is the physical address for when getting a chunk from the counter.

Storage of a free list big enough to store all physical block addresses.

Maintain three pointers for the free list that are reset to zero. The first one we will call write_ptr. This pointer will identify the next location to write the physical address of a block to be de-allocated. Note: we can never free more physical memory locations than we have. Once recording address the pointer will be incremented to walk the free list like a ring.

The second pointer will be called stop_ptr. The stop_ptr pointer will be advanced by the number of address chunks de-allocates when a context finishes. The address between the stop_ptr and write_ptr cannot be reused because they are still in use. But as soon as the context using then is dismissed the stop_ptr will be advanced.

The third pointer will be called read_ptr. This pointer will point will point to the next address that can be used for allocation as long as the read_ptr does not equal the stop_ptr and the IFC is at its maximum count.

5.3.5 *De-allocate Block*

This block will maintain a free physical address block count for each context. While in current context, a count shall be maintained specifying how many blocks were written into the free list at the write_ptr pointer. This count will be reset upon reset or when this context is active on the back and different than the previous context. It is actually a count of blocks in the previous context that will no longer be used. This count will be used to advance the write_ptr pointer to make available the set of physical blocks freed when the previous context was done. This allows the discard or de-allocation of any number of blocks in one clock.

5.3.6 *Operation of Incremental model*

The basic operation of the model would start with the write_ptr, stop_ptr, read_ptr pointers in the free list set to zero and the free list counter is set to zero. Also all the dirty bits and the previous context will be initialized to zero. When the first set constants happen, the reset dirty bit will not be set, so we will allocate a physical location from the free list counter because its not at the max value. The data will be written into physical address zero. Both the additional copy of the renaming table and the context zeros of the big renaming table will be updated for the logical address that was written by set start with physical address of 0. This process will be repeated for any logical address that are not dirty until the context changes. If a logical address is hit that has its dirty bits set while in the same context, both dirty bits would be set, so the new data will be over-written to the last physical address assigned for this logical address. When the first draw command of the context is detected, the previous context stored in the additional renaming table will be copied to the larger renaming table in the current (new) context location. Then the set constant logical address with be loaded with a new physical address during the copy and if the reset dirty was set, the physical address it replaced in the renaming table would be entered at the write_ptr pointer location on the free list and the write_ptr will be incremented. The de-allocation counter for the previous context (eight) will be incremented. This as set states come in for this context one of the following will happen:

1.) No dirty bits are set for the logical address being updated. A line will be allocated of the free-list counter or the free list at read_ptr pointer if read_ptr != to stop_ptr .
2.) Reset dirty set and Context dirty not set. A new physical address is allocated, the physical address in the renaming table is put on the free list at write_ptr and it is incremented along with the de-allocate counter for the last context.
3.) Context dirty is set then the data will be written into the physical address specified by the logical address.

This process will continue as long as set states arrive. This block will provide backpressure to the CP whenever he has not free list entries available (counter at max and stop_ptr == read_ptr). The command stream will keep a count of contexts of constants in use and prevent more than max constants contexts from being sent.

Whenever a draw packet arrives, the content of the re-mapping table is written to the correct re-mapping table for the context number. Also if the next context uses less constants than the current one all exceeding lines are moved to the free list to be de-allocated later. This happens in parallel with the writing of the re-mapping table to the correct memory.

Now preferable when the constant context leaves the last ALU clause it will be sent to this block and compared with the previous context that left. (Init to zero) If they differ than the older context will no longer be referenced and thus can be de-allocated in the physical memory. This is accomplished by adding the number of blocks freed this context to the stop_ptr pointer. This will make all the physical addresses used by this context available to the read_ptr allocate pointer for future allocation.

This device allows representation of multiple contexts of constants data with N copies of the logical address space. It also allows the second context to be represented as the first set plus some new additional data by just storing the delta's. It allows memory to be efficiently used and when the constants updates are small it can store multiple context. However, if the updates are large, less contexts will be stored and potentially performance will be degraded. Although it will still perform as well as a ring could in this case.

5.4 Constant Store Indexing

In order to do constant store indexing, the sequencer must be loaded first with the indexes (that come from the GPRs). There are 144 wires from the exit of the SP to the sequencer (9 bits pointers x 16 vertexes/clock). Since the data must pass thru the Shader pipe for the float to fixed conversion, there is a latency of 4 clocks (1 instruction)

between the time the sequencer is loaded and the time one can index into the constant store. The assembly will look like this

```
MOVA  R1.X,R2.X       // Loads the sequencer with the content of R2.X, also copies the content of R2.X into R1.X
NOP                   // latency of the float to fixed conversion
ADD   R3,R4,C0[R2.X]// Uses the state from the sequencer to add R4 to C0[R2.X] into R3
```

Note that we don't really care about what is in the brackets because we use the state from the MOVA instruction. R2.X is just written again for the sake of simplicity and coherency.

The storage needed in the sequencer in order to support this feature is 2*64*9 bits = 1152 bits.

5.5 Real Time Commands

The real time commands constants are written by the CP using the register mapped registers allocated for RT. It works is the same way than when dealing with regular constant loads BUT in this case the CP is not sending a logical address but rather a physical address and the reads are not passing thru the re-mapping table but are directly read from the memory. The boundary between the two zones is defined by the CONST_EO_RT control register. Similarly, for the fetch state, the boundary between the two zones is defined by the TSTATE_EO_RT control register.

5.6 Constant Waterfalling

In order to have a reasonable performance in the case of constant store indexing using the address register, we are going to have the possibility of using the physical memory port for read only. This way we can read 1 constant per clock and thus have a worst-case waterfall mode of 1 vertex per clock. There is a small synchronization issue related with this as we need for the SQ to make sure that the constants where actually written to memory (not only sent to the sequencer) before it can allow the first vector of pixels or vertices of the state to go thru the ALUs. To do so, the sequencer keeps 8 bits (one per render state) and sets the bits whenever the last render state is written to memory and clears the bit whenever a state is freed.

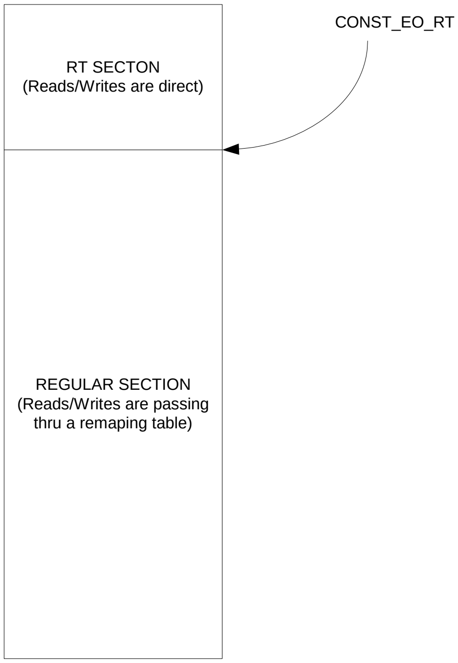


Figure 9: The Constant store

6. Looping and Branches

Loops and branches are planned to be supported and will have to be dealt with at the sequencer level. We plan on supporting constant loops and branches using a control program.

6.1 The controlling state.

The R400 controling state consists of:

Boolean[256:0]
Loop_count[7:0][31:0]
Loop_Start[7:0][31:0]
Loop_Step[7:0][31:0]

That is 256 Booleans and 32 loops.

We have a stack of 4 elements for nested calls of subroutines and 4 loop counters to allow for nested loops.

This state is available on a per shader program basis.

6.2 The Control Flow Program

We'd like to be able to code up a program of the form:

```
1:      Loop
2:      Exec    TexFetch
3:              TexFetch
4:              ALU
5:              ALU
6:              TexFetch
7:      End Loop
8:      ALU Export
```

But realize that 3: may be dependent on 2: and 4: is almost certainly dependent on 2: and 3:. Without clausing, these dependencies need to be expressed in the Control Flow instructions. Additionally, without separate 'texture clauses' and 'ALU clauses' we need to know which instructions to dispatch to the Texture Unit and which to the ALU unit. This information will be encapsulated in the flow control instructions.

Each control flow instruction will contain 2 bits of information for each (non-control flow) instruction:

a) ALU or Texture
b) Serialize Execution

(b) would force the thread to stop execution at this point (before the instruction is executed) and wait until all textures have been fetched. Given the allocation of reserved bits, this would mean that the count of an 'Exec' instruction would be limited to about 8 (non-control-flow) instructions. If more than this were needed, a second Exec (with the same conditions) would be issued.

Another function that relies upon 'clauses' is allocation and order of execution. We need to assure that pixels and vertices are exported in the correct order (even if not all execution is ordered) and that space in the output buffers are allocated in order. Additionally data can't be exported until space is allocated. A new control flow instruction:

Alloc <buffer select -- position,parameter, pixel or vertex memory. And the size required>.

would be created to mark where such allocation needs to be done. To assure allocation is done in order, the actual allocation for a given thread can not be performed unless the equivalent allocation for all previous threads is already completed. The implementation would also assure that execution of instruction(s) following the serialization due to the Alloc will occur in order -- at least until the next serialization or change from ALU to Texture. In most cases this will allow the exports to occur without any further synchronization. Only 'final' allocations or position allocations are

guaranteed to be ordered. Because strict ordering is required for pixels, parameters and positions, this implies only a single alloc for these structures. Vertex exports to memory do not require ordering during allocation and so multiple 'allocs' may be done.

6.2.1 *Control flow instructions table*

Here is the revised control flow instruction set.

Note that whenever a field is marked as RESERVED, it is assumed that all the bits of the field are cleared (0).

NOP		
47 … 44	43	42 … 0
0000	Addressing	RESERVED

This is a regular NOP.

Execute					
47 … 44	43	40 … 34	33 …16	15…12	11 … 0
0001	Addressing	RESERVED	Instructions type + serialize (9 instructions)	Count	Exec Address

Execute_End					
47 … 44	43	40 … 34	33 …16	15…12	11 … 0
0010	Addressing	RESERVED	Instructions type + serialize (9 instructions)	Count	Exec Address

Execute up to 9 instructions at the specified address in the instruction memory. The Instruction type field tells the sequencer the type of the instruction (LSB) (1 = Texture, 0 = ALU and whether to serialize or not the execution (MSB) (1 = Serialize, 0 = Non-Serialized). If Execute_End this is the last execution block of the shader program.

Conditional_Execute						
47 … 44	43	42	41 … 34	33…16	15 …12	11 … 0
0011	Addressing	Condition	Boolean address	Instructions type + serialize (9 instructions)	Count	Exec Address

Conditional_Execute_End						
47 … 44	43	42	41 … 34	33…16	15 …12	11 … 0
0100	Addressing	Condition	Boolean address	Instructions type + serialize (9 instructions)	Count	Exec Address

If the specified Boolean (8 bits can address 256 Booleans) meets the specified condition then execute the specified instructions (up to 9 instructions). If the condition is not met, we go on to the next control flow instruction. If Conditional_Execute_End and the condition is met, this is the last execution block of the shader program.

Conditional_Execute_Predicates							
47 … 44	43	42	41 … 36	35 … 34	33…16	15…12	11 … 0
0101	Addressing	Condition	RESERVED	Predicate vector	Instructions type + serialize (9 instructions)	Count	Exec Address

Conditional_Execute_Predicates_End							
47 … 44	43	42	41 … 36	35 … 34	33…16	15…12	11 … 0
0110	Addressing	Condition	RESERVED	Predicate vector	Instructions type + serialize (9 instructions)	Count	Exec Address

Check the AND/OR of all current predicate bits. If AND/OR matches the condition execute the specified number of instructions. We need to AND/OR this with the kill mask in order not to consider the pixels that aren’t valid. If the

condition is not met, we go on to the next control flow instruction. If Conditional_Execute_Predicates_End and the condition is met, this is the last execution block of the shader program.

Conditional_Execute_Predicates_No_Stall							
47 … 44	43	42	41 … 36	35 … 34	33…16	15…12	11 … 0
1101	Addressing	Condition	RESERVED	Predicate vector	Instructions type + serialize (9 instructions)	Count	Exec Address

Conditional_Execute_Predicates_No_Stall_End							
47 … 44	43	42	41 … 36	35 … 34	33…16	15…12	11 … 0
1110	Addressing	Condition	RESERVED	Predicate vector	Instructions type + serialize (9 instructions)	Count	Exec Address

Same as Conditionnal_Execute_Predicates but the SQ is not going to wait for the predicate vector to be updated. You can only set this in the compiler if you know that the predicate set is only a refinement of the current one (like a nested if) because the optimization would still work.

Loop_Start					
47 … 44	43	42 … 21	20 … 16	15…12	11 … 0
0111	Addressing	RESERVED	loop ID	RESERVED	Jump address

Loop Start. Compares the loop iterator with the end value. If loop condition not met jump to the address. Forward jump only. Also computes the index value. The loop id must match between the start to end, and also indicates which control flow constants should be used with the loop.

Loop_End						
47 … 44	43	42 … 24	23… 21	20 … 16	15…12	11 … 0
1000	Addressing	RESERVED	Predicate break	loop ID	RESERVED	start address

Loop end. Increments the counter by one, compares the loop count with the end value. If loop condition met, continue, else, jump BACK to the start of the loop. If predicate break != 0, then compares predicate vector n (specified by predicate break number). If all bits cleared then break the loop.

The way this is described does not prevent nested loops, and the inclusion of the loop id make this easy to do.

Conditionnal_Call						
47 … 44	43	42	41 … 34	33 … 13	12	11 … 0
1001	Addressing	Condition	Boolean address	RESERVED	Force Call	Jump address

If the condition is met, jumps to the specified address and pushes the control flow program counter on the stack. If force call is set the condition is ignored and the call is made always.

Return		
47 … 44	43	42 … 0
1010	Addressing	RESERVED

Pops the topmost address from the stack and jumps to that address. If nothing is on the stack, the program will just continue to the next instruction.

Conditionnal_Jump							
47 … 44	43	42	41… 34	33	32 … 13	12	11 … 0
1011	Addressing	Condition	Boolean address	FW only	RESERVED	Force Jump	Jump address

If force jump is set the condition is ignored and the jump is made always. If FW only is set then only forward jumps are allowed.

Allocate				
47 … 44	43	42…41	40 … 3	2…0
1100	Debug	Buffer Select	RESERVED	Size

Buffer Select takes a value of the following:
01 – position export (ordered export)
10 – parameter cache or pixel export (ordered export)
11 – pass thru (out of order exports).

Size field is only used to reserve space in the export buffer for pass thru exports. Valid values are 1 (1 line) thru 9 (9 lines). It should be determined by the compiler/assembler by taking max index used +1.

If debug is set this is a debug alloc (ignore if debug DB_ON register is set to off).

6.3 Implementation

The envisioned implementation has a buffer that maintains the state of each thread. A thread lives in a given location in the buffer during its entire life, but the buffer has FIFO qualities in that threads leave in the order that they enter. Actually two buffers are maintained -- one for Vertices and one for Pixels. The intended implementation would allow for:

16 entries for vertices
48 entries for pixels.

From each buffer, arbitration logic attempts to select 1 thread for the texture unit and 1 (interleaved) thread for the ALU unit. Once a thread is selected it is read out of the buffer, marked as invalid, and submitted to appropriate execution unit. It is returned to the buffer (at the same place) with its status updated once all possible sequential instructions have been executed. A switch from ALU to TEX or visa-versa or a Serialize_Execution modifier forces the thread to be returned to the buffer.

Each entry in the buffer will be stored across two physical pieces of memory - most bits will be stored in a 1 read port device. Only bits needed for thread arbitration will be stored in a highly multi-ported structure. The bits kept in the 1 read port device will be termed 'state'. The bits kept in the multi-read ported device will be termed 'status'.

'State Bits' needed include:

1. Control Flow Instruction Pointer (13 bits),
2. Execution Count Marker 4 bits),
3. Loop Iterators (4x9 bits),
4. Call return pointers (4x12 bits),
5. Predicate Bits (64 bits),
6. Export ID (1 bit),
7. Parameter Cache base Ptr (7 bits),
8. GPR Base Ptr (8 bits),
9. Context Ptr (3 bits).
10. LOD corrections (6x16 bits)
11. Valid bits (64 bits)
12. RT (1 bit) Signifies that this thread is a Real Time thread. This bit must be sent to the Constant store state machine when reading it.

Absent from this list are 'Index' pointers. These are costly enough that I'm presuming that they are instead stored in the GPRs. The first seven fields above (Control Flow Ptr, Execution Count, Loop Counts, call return ptrs, Predicate bits, PC base ptr and export ID) are updated every time the thread is returned to the buffer based on how much progress has been mode on thread execution. GPR Base Ptr, Context Ptr and LOD corrections are unchanged throughout execution of the thread.

'Status Bits' needed include:

- Valid Thread
- Texture/ALU engine needed
- Texture Reads are outstanding
- Waiting on Texture Read to Complete
- Allocation Wait (2 bits)
- 00 – No allocation needed
- 01 – Position export allocation needed (ordered export)
- 10 – Parameter or pixel export needed (ordered export)
- 11 – pass thru (out of order export)
- Allocation Size (4 bits)
- Position Allocated
- Mem/Color Allocated
- First thread of a new context
- Event thread (NULL thread that needs to trickle down the pipe)
- Last (1 bit)
- Pulse SX (1 bit)

Formatted: Bullets and Numbering

All of the above fields from all of the entries go into the arbitration circuitry. The arbitration circuitry will select a winner for both the Texture Engine and for the ALU engine. There are actually two sets of arbitration -- one for pixels and one for vertices. A final selection is then done between the two. But the rest of this implementation summary only considers the 'first' level selection which is similar for both pixels and vertices.

Texture arbitration requires no allocation or ordering so it is purely based on selecting the 'oldest' thread that requires the Texture Engine.

ALU arbitration is a little more complicated. First, only threads where either of Texture_Reads_outstanding or Waiting_on_Texture_Read_to_Complete are '0' are considered. Then if Allocation_Wait is active, these threads are further filtered based on whether space is available. If the allocation is position allocation, then the thread is only considered if all 'older' threads have already done their position allocation (position allocated bits set). If the allocation is parameter or pixel allocation, then the thread is only considered if it is the oldest thread. Also a thread is not considered if it is a parameter or pixel or position allocation, has its First_thread_of_a_new_context bit set and would cause ALU interleaving with another thread performing the same parameter or pixel or position allocation. Finally the 'oldest' of the threads that pass through the above filters is selected. If the thread needed to allocate, then at this time the allocation is done, based on Allocation_Size. If a thread has its “last” bit set, then it is also removed from the buffer, never to return.

If I now redefine 'clauses' to mean 'how many times the thread is removed from the thread buffer for the purpose of exection by either the ALU or Texture engine', then the minimum number of clauses needed is 2 -- one to perform the allocation for exports (execution automatically halts after an 'Alloc' instruction) (but doesn't performs the actual allocation) and one for the actual ALU/export instructions. As the 'Alloc' instruction could be part of a texture clause (presumably the final instruction in such a clause), a thread could still execute in this minimal number of 2 clauses, even if it involved texture fetching.

The Texture_Reads_Outstanding bit must be updated by the sequencer, based on keeping track of how many Texture Clauses have been executed by a given thread that have not yet had there data returned. Any number above 0 results in this bit being set. We could consider forcing synchronization such that two texture clauses for a given thread may not be outstanding at any time (that would be my preference for simplicity reasons and because it would require only very little change in the texture pipe interface). This would allow the sequencer to set the bit on execution of the texture clause, and allow the texture unit to return a pointer to the thread buffer on completion that clears the bit.

6.4 Data dependant predicate instructions

Data dependant conditionals will be supported in the R400. The only way we plan to support those is by supporting three vector/scalar predicate operations of the form:

PRED_SETE_# - similar to SETE except that the result is 'exported' to the sequencer.
PRED_SETNE_# - similar to SETNE except that the result is 'exported' to the sequencer.
PRED_SETGT_# - similar to SETGT except that the result is 'exported' to the sequencer
PRED_SETGTE_# - similar to SETGTE except that the result is 'exported' to the sequencer

For the scalar operations only we will also support the two following instructions:
PRED_SETE0_# – SETE0
PRED_SETE1_# – SETE1

The export is a single bit - 1 or 0 that is sent using the same data path as the MOVA instruction. The sequencer will maintain 4 sets of 64 bit predicate vectors (in fact 8 sets because we interleave two programs but only 4 will be exposed) and use it to control the write masking. This predicate is not maintained across clause boundaries. The # sign is used to specify which predicate set you want to use 0 thru 3.

Then we have two conditional execute bits. The first bit is a conditional execute “on” bit and the second bit tells us if we execute on 1 or 0. For example, the instruction:

P0_ADD_# R0,R1,R2

Is only going to write the result of the ADD into those GPRs whose predicate bit is 0. Alternatively, P1_ADD_# would only write the results to the GPRs whose predicate bit is set. The use of the P0 or P1 without precharging the sequencer with a PRED instruction is undefined.

{Issue: do we have to have a NOP between PRED and the first instruction that uses a predicate?}

6.5 HW Detection of PV,PS

Because of the control program, the compiler cannot detect statically dependant instructions. In the case of non-masked writes and subsequent reads the sequencer will insert uses of PV,PS as needed. This will be done by comparing the read address and the write address of consecutive instructions. For masked writes, the sequencer will insert NOPs wherever there is a dependant read/write.

The sequencer will also have to insert NOPs between PRED_SET and MOVA instructions and their uses.

6.6 Register file indexing

Because we can have loops in fetch clause, we need to be able to index into the register file in order to retrieve the data created in a fetch clause loop and use it into an ALU clause. The instruction will include the base address for register indexing and the instruction will contain these controls:

Bit7	Bit 6	
0	0	'absolute register'
0	1	'relative register'
1	0	'previous vector'
1	1	'previous scalar'

In the case of an absolute register we just take the address as is. In the case of a relative register read we take the base address and we add to it the loop_index and this becomes our new address that we give to the shader pipe.

The sequencer is going to keep a loop index computed as such:

Index = Loop_iterator*Loop_step + Loop_start.

We loop until loop_iterator = loop_count. Loop_step is a signed value [-128…127]. The computed index value is a 10 bit counter that is also signed. Its real range is [-256,256]. The tenth bit is only there so that we can provide an out of range value to the “indexing logic” so that it knows when the provided index is out of range and thus can make the necessary arrangements.

6.7 Debugging the Shaders

In order to be able to debug the pixel/vertex shaders efficiently, we provide 2 methods.

6.7.1 *Method 1: Debugging registers*

Current plans are to expose 2 debugging, or error notification, registers:
1. address register where the first error occurred
2. count of the number of errors

The sequencer will detect the following groups of errors:
- count overflow
- constant indexing overflow
- register indexing overflow

Compiler recognizable errors:
- jump errors
  relative jump address > size of the control flow program
- call stack
  call with stack full
  return with stack empty

A jump error will always cause the program to break. In this case, a break means that a clause will halt execution, but allowing further clauses to be executed.

With all the other errors, program can continue to run, potentially to worst-case limits. The program will only break if the DB_PROB_BREAK register is set.

If indexing outside of the constant or the register range, causing an overflow error, the hardware is specified to return the value with an index of 0. This could be exploited to generate error tokens, by reserving and initializing the 0th register (or constant) for errors.

{ISSUE : Interrupt to the driver or not?}

6.7.2 *Method 2: Exporting the values in the GPRs*

1) The sequencer will have a debug active, count register and an address register for this mode.

Under the normal mode execution follows the normal course.

Under the debug mode it is assumed that the program is always exporting n debug vectors and that all other exports to the SX block (position, color, z, ect) will been turned off (changed into NOPs) by the sequencer (even if they occur before the address stated by the ADDR debug register).

7. Pixel Kill Mask

A vector of 64 bits is kept by the sequencer per group of pixels/vertices. Its purpose is to optimize the texture fetch requests and allow the shader pipe to kill pixels using the following instructions:

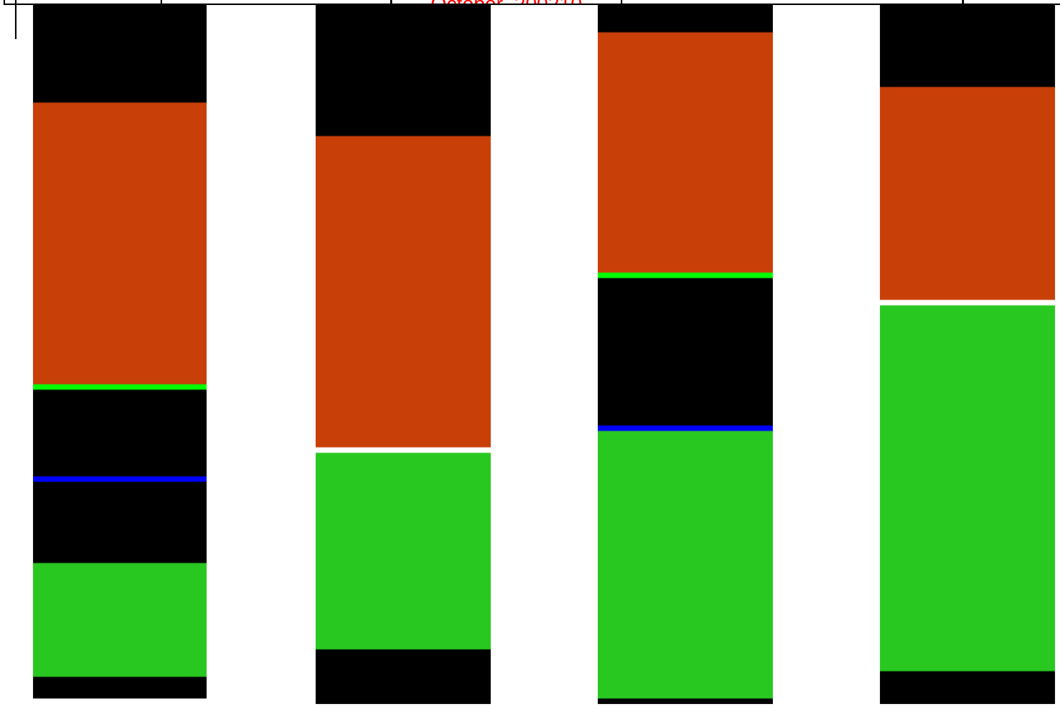
MASK_SETE
MASK_SETNE
MASK_SETGT
MASK_SETGTE

8. Multipass vertex shaders (HOS)

Multipass vertex shaders are able to export from the 6 last clauses but to memory ONLY.

9. Register file allocation

The register file allocation for vertices and pixels can either be static or dynamic. In both cases, the register file in managed using two round robins (one for pixels and one for vertices). In the dynamic case the boundary between pixels and vertices is allowed to move, in the static case it is fixed to 128-VERTEX_REG_SIZE for vertices and PIXEL_REG_SIZE for pixels.



Above is an example of how the algorithm works. Vertices come in from top to bottom; pixels come in from bottom to top. Vertices are in orange and pixels in green. The blue line is the tail of the vertices and the green line is the tail of the pixels. Thus anything between the two lines is shared. When pixels meets vertices the line turns white and the boundary is static until both vertices and pixels share the same “unallocated bubble”. Then the boundary is allowed to move again. The numbering of the GPRs starts from the bottom of the picture at index 0 and goes up to the top at index 127.

10. Fetch Arbitration

The fetch arbitration logic chooses one of the n potentially pending fetch clauses to be executed. The choice is made by looking at the Vs and Ps reservation stations and picking the first one ready to execute. Once chosen, the clause state machine will send one 2x2 fetch per clock (or 4 fetches in one clock every 4 clocks) until all the fetch instructions of the clause are sent. This means that there cannot be any dependencies between two fetches of the same clause.

The arbitrator will not wait for the fetches to return prior to selecting another clause for execution. The fetch pipe will be able to handle up to X(?) in flight fetches and thus there can be a fair number of active clauses waiting for their fetch return data.

11. ALU Arbitration

ALU arbitration proceeds in almost the same way than fetch arbitration. The ALU arbitration logic chooses one of the n potentially pending ALU clauses to be executed. The choice is made by looking at the Vs and Ps reservation stations and picking the first one ready to execute. There are two ALU arbiters, one for the even clocks and one for the odd clocks. For example, here is the sequencing of two interleaved ALU clauses (E and O stands for Even and Odd sets of 4 clocks):

Einst0 Oinst0 Einst1 Oinst1 Einst2 Oinst2 Einst0 Oinst3 Einst1 Oinst4 Einst2 Oinst0…

Proceeding this way hides the latency of 8 clocks of the ALUs. Also note that the interleaving also occurs across clause boundaries.

12. Handling Stalls

When the output file is full, the sequencer prevents the ALU arbitration logic from selecting the last clause (this way nothing can exit the shader pipe until there is place in the output file. If the packet is a vertex packet and the position buffer is full (POS_FULL) then the sequencer also prevents a thread from entering an exporting clause. The sequencer will set the OUT_FILE_FULL signal n clocks before the output file is actually full and thus the ALU arbiter will be able read this signal and act accordingly by not preventing exporting clauses to proceed.

13. Content of the reservation station FIFOs

The reservation FIFOs contain the state of the vector of pixels and vertices. We have two sets of those: one for pixels, and one for vertices. They contain 3 bits of Render State 7 bits for the base address of the GPRs, some bits for LOD correction and coverage mask information in order to fetch fetch for only valid pixels, the quad address.

14. The Output File

The output file is where pixels are put before they go to the RBs. The write BW to this store is 256 bits/clock. Just before this output file are staging registers with write BW 512 bits/clock and read BW 256 bits/clock. The staging registers are 4x128 (and there are 16 of those on the whole chip).

15. IJ Format

The IJ information sent by the PA is of this format on a per quad basis:

We have a vector of IJ's (one IJ per pixel at the centroid of the fragment or at the center of the pixel depending on the mode bit). All pixel's parameters are always interpolated at full 20x24 mantissa precision.

$$
\begin{aligned}
P0 &= A + I(0)*(B-A) + J(0)*(C-A) \\
P1 &= A + I(1)*(B-A) + J(1)*(C-A) \\
P2 &= A + I(2)*(B-A) + J(2)*(C-A) \\
P3 &= A + I(3)*(B-A) + J(3)*(C-A)
\end{aligned}
$$

P0	P1
P2	P3

Multiplies (Full Precision): 8
Subtracts 19x24 (Parameters): 2
Adds: 8

FORMAT OF P's IJ : Mantissa 20 Exp 4 for I + Sign
Mantissa 20 Exp 4 for J + Sign

Total number of bits : 20*8 + 4*8 + 4*2 = 200.

All numbers are kept using the un-normalized floating point convention: if exponent is different than 0 the number is normalized if not, then the number is un-normalized. The maximum range for the IJs (Full precision) is +/- 1024.

15.1 Interpolation of constant attributes

Because of the floating point imprecision, we need to take special provisions if all the interpolated terms are the same or if two of the terms are the same.

16. Staging Registers

In order for the reuse of the vertices to be 14, the sequencer will have to re-order the data sent IN ORDER by the VGT for it to be aligned with the parameter cache memory arrangement. Given the following group of vertices sent by the VGT:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 || 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 || 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 || 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63

The sequencer will re-arrange them in this fashion:

0 1 2 3 16 17 18 19 32 33 34 35 48 49 50 51 || 4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55 || 8 9 10 11 24 25 26 27 40 41 42 43 56 57 58 59 || 12 13 14 15 28 29 30 31 44 45 46 47 60 61 62 63

The || markers show the SP divisions. In the event a shader pipe is broken, the SQ is responsible to insert padding to account for the missing pipe. For example, if SP1 is broken, vertices 4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55 will not be sent by the VGT to the SQ **AND** the SQ is responsible to “jump” over these vertices in order for no valid vertices to be sent to an invalid SP.

The most straightforward, *non-compressed* interface method would be to convert, in the VGT, the data to 32-bit floating point prior to transmission to the VSISRs. In this scenario, the data would be transmitted to (and stored in) the VSISRs in full 32-bit floating point. This method requires three 24-bit fixed-to-float converters in the VGT. Unfortunately, it also requires and additional 3,072 bits of storage across the VSISRs. This interface is illustrated in Figure 11~~Figure 11Figure 11~~. The area of the fixed-to-float converters and the VSISRs for this method is roughly estimated as 0.759sqmm using the R300 process. The gate count estimate is shown in Figure 10~~Figure 10Figure 10~~.

Basis for 8-deep Latch Memory (from R300)			
8x24-bit	11631 μ^2		60.57813 μ^2 per bit
Area of 96x8-deep Latch Memory	46524 μ^2		
Area of 24-bit Fix-to-float Converter	4712 μ^2 per converter		
Method 1	Block	Quantity	Area
	F2F	3	14136
	8x96 Latch	16	744384
			758520 μ^2

Figure 10:Area Estimate for VGT to Shader Interface

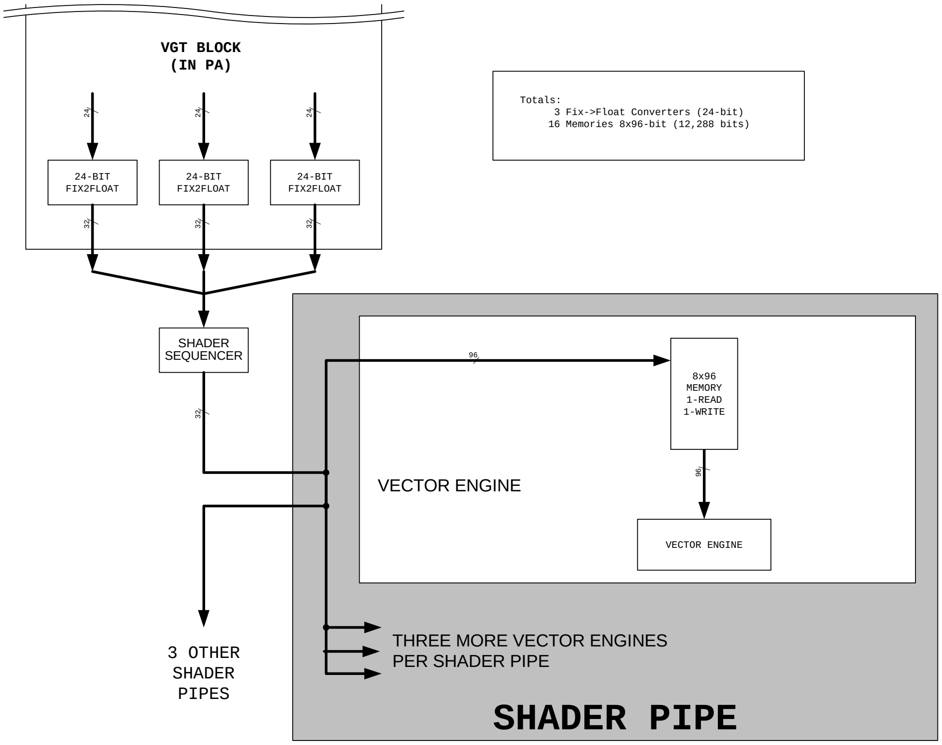


Figure 11:VGT to Shader Interface

17. The parameter cache

The parameter cache is where the vertex shaders export their data. It consists of 16 128x128 memories (1R/1W). The reuse engine will make it so that all vertexes of a given primitive will hit different memories. The allocation method for these memories is a simple round robin. The parameter cache pointers are mapped in the following way: 4MSBs are the memory number and the 7 LSBs are the address within this memory.

MEMORY NUMBER 4 bits	ADDRESS 7 bits

The PA generates the parameter cache addresses as the positions come from the SQ. All it needs to do is keep a Current_Location pointer (7 bits only) and as the positions comes increment the memory number. When the memory number field wraps around, the PA increments the Current_Location by VS_EXPORT_COUNT (a snooped register from the SQ). As an example, say the memories are all empty to begin with and the vertex shader is exporting 8 parameters per vertex (VS_EXPORT_COUNT = 8). The first position received is going to have the PC address 00000000000 the second one 00010000000, third one 00100000000 and so on up to 11110000000. Then the next position received (the 17th) is going to have the address 00000001000, the 18th 00010001000, the 19th 00100001000 and so on. The Current_location is NEVER reset BUT on chip resets. The only thing to be careful about is that if the SX doesn't send you a full group of positions (<64) then you need to fill the address space so that the next group starts correctly aligned (for example if you receive only 33 positions then you need to add 2*VS_EXPORT_COUNT to Current_Location and reset the memory count to 0 before the next vector begins).

17.1 Export restrictions

17.1.1 *Pixel exports:*

Pixels can export 1,2,3 or 4 color buffers to the SX(+z). The exports will be done in order. The PRED_OPTIMIZE function has to be turned of if the exports are done using interleaved predicated instructions. The exports will always be ordered to the SX.

17.1.2 *Vertex exports:*

Position or parameter caches can be exported in any order in the shader program. It is always better to export posistion as soon as possible. Position has to be exported in a single export block (no texture instructions can be placed between the exports). Parameter cache exports can be done in any order with texture instructions interleaved. The PRED_OPTIMIZE function has to be turned of if the exports are done using interleaved predicated instructions to the Parameter cache (see Arbitration restrictions for details). The exports will always be allocated in order to the SX.

17.1.3 *Pass thru exports:*

Pass thru exports have to be done in groups of the form:

```
Alloc 4 (8 or 12)
Execute ALU(ADDR) ALU(DATA) ALU(DATA) ALU(DATA)…
```

They cannot have texture instructions interleaved in the export block. These exports **are not guaranteed to be ordered**.

Also, when doing a pass thru export, Position MUST be exported AFTER all pass thru exports. This position export is used to synchronize the chip when doing a transition from pass thru shader to regular shader and vice versa.

17.2 Arbitration restrictions

Here are the Sequencer arbitration restrictions:

1) Cannot execute a serialized thread if the corresponding texture pending bit is set
2) Cannot allocate position if any older thread has not allocated position
3) If last thread is marked as not valid AND marked as last and we are about to execute the second to oldest thread also marked last then:
    a. Both threads must be from the same context (cannot allow a first thread)
    b. Must turn off the predicate optimization for the second thread
4) Cannot execute a texture clause if texture reads are pending
5) Cannot execute last if texture pending (even if not serial)

18. Export Types

The export type (or the location where the data should be put) is specified using the destination address field in the ALU instruction. Here is a list of all possible export modes:

18.1 Vertex Shading

0:15	- 16 parameter cache
16:31	- Empty (Reserved?)
32	- Export Address
33:~~41~~ 37	- ~~9~~ 5 vertex exports to the frame buffer and index
~~42~~38:47	- Empty
48:~~55~~52~~5~~	- ~~8~~ 5 debug export (interpret as normal ~~vertex~~ memory export)
60	- export addressing mode
61	- Empty
62	- position

63 - sprite size export that goes with position export
(X= point size, Y= edge flag is bit 0, Z= VtxKill is bitwise OR of bits 30:0. Any bit other than sign means VtxKill.)

18.2 Pixel Shading

0	- Color for buffer 0 (primary)
1	- Color for buffer 1
2	- Color for buffer 2
3	- Color for buffer 3
4:15	- Empty
16	- Buffer 0 Color/Fog (primary)
17	- Buffer 1 Color/Fog
18	- Buffer 2 Color/Fog
19	- Buffer 3 Color/Fog
20:31	- Empty
32	- Export Address
33:~~41~~37	- ~~9~~ 5 exports for multipass pixel shaders.
~~42~~38:47	- Empty
48:~~5~~52~~5~~	- ~~8~~5 debug exports (interpret as normal ~~pixel~~ memory export)
60	- export addressing mode
61	- Z for primary buffer (Z exported to 'alpha' component)
62:63	- Empty

19. Special Interpolation modes

19.1 Real time commands

We are unable to use the parameter memory since there is no way for a command stream to write into it. Instead we need to add three 16x128 memories (one for each of three vertices x 16 interpolants). These will be mapped onto the register bus and written by type 0 packets, and output to the the parameter busses (the sequencer and/or PA need to be able to address the realtime parameter memory as well as the regular parameter store. For higher performance we should be able able to view them as two banks of 16 and do double buffering allowing one to be loaded, while the other is rasterized with. Most overlay shaders will need 2 or 4 scalar coordinates, one option might be to restrict the memory to 16x64 or 32x64 allowing only two interpolated scalars per cycle, the only problem I see with this is, if we view support for 16 vector-4 interpolants important (true only if we map Microsoft's high priority stream to the realtime stream), then the PA/sequencer need to support a realtime-specific mode where we need to address 32 vectors of parameters instead of 16. This mode is triggered by the primitive type: REAL TIME. The actual memories are in the in the SX blocks. The parameter data memories are hooked on the RBBM bus and are loaded by the CP using register mapped memory.

19.2 Sprites/ XY screen coordinates/ FB information

XY screen coordinates may be needed in the shader program. This functionality is controlled by the param_gen_I0 register (in SQ) in conjunction with the SND_XY register (in SC) and the param_gen_pos. Also it is possible to send the faceness information (for OGL front/back special operations) to the shader using the same control register. Here is a list of all the modes and how they interact together:

The Data is going to be written in the register specified by the param_gen_pos register.

Param_Gen_I0 disable, snd_xy disable = No modification
Param_Gen_I0 disable, snd_xy enable = No modification
Param_Gen_I0 enable, snd_xy disable = Sign(faceness)garbage,(Sign Point)garbage,Sign(Line)s, t
Param_Gen_I0 enable, snd_xy enable = Sign(faceness)screenX,(Sign Point)screenY,Sign(Line)s, t

In other words,
The generated vector is (X in RED, Y in GREEN, S in BLUE and T in ALPHA):
X,Y,S,T

These values are always supposed to be positive and any shader use of them should use the ABS function (as their sign bits will now be used for flags).
SignX = BackFacing
SignY = Point Primitive
SignS = Line Primitive
SignT = currently unused as a flag.

If !Point & !Line, then it is a Poly.

I would assume that one implementation which allows for generic texture lookup (using 3D maps) for poly stipple and AA for the driver would be

```
if(Y<0) {
        R = 0.0 (Point)
} else if (S < 0) {
        R = 1.0 (Line)
} else {
        R = 2.0 (Poly)
}
```

19.3 Auto generated counters

In the cases we are dealing with multipass shaders, the sequencer is going to generate a vector count to be able to both use this count to write the 1st pass data to memory and then use the count to retrieve the data on the 2nd pass. The count is always generated in the same way but it is passed to the shader in a slightly different way depending on the shader type (pixel or vertex). This is toggled on and off using the GEN_INDEX_PIX/VTX register. The sequencer is going to keep two counters, one for pixels and one for vertices. Every time a full vector of vertices or pixels is written to the GPRs the counter is incremented. Every time a RST_PIX_COUNT or RST_VTX_COUNT events are received, the corresponding counter is reset. While there is only one count broadcast to the GPRs, the LSB are hardwired to specific values making the index different for all elements in the vector. Since the count must be different for all pixels/vertices and the 4 LSBs (16 positions) are hardwired to the corresponding shader unit the SQ has two choices:

1) Maintain a 19 bit counter that counts the vectors of 64. In this case the phase must be appended to the count before the count is broadcast to the SPs:

Counter (19 bits)	Phase (2 bits)	Hardwired (4 bits)

2) Maintain a 21 bits counter that counts sub-vectors of 16. In this case only the counter is sent to the Sps:

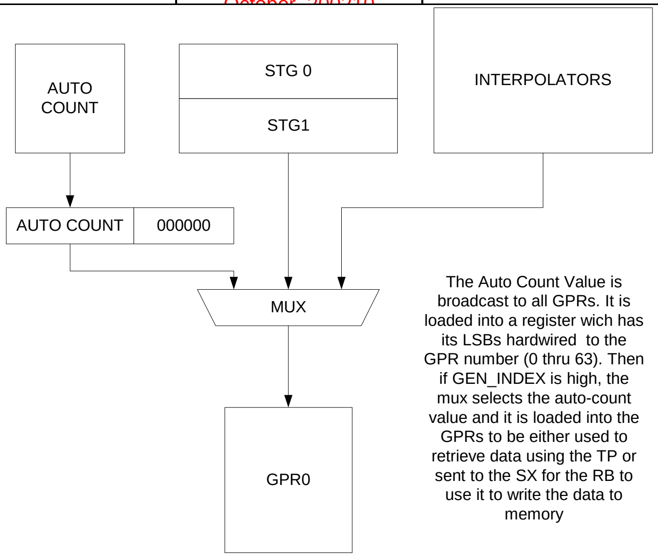
Counter (21 bits)	~~Harwired~~Hardwired (4 bits)

19.3.1 *Vertex shaders*

In the case of vertex shaders, if GEN_INDEX_VTX is set, the data will be put into the x field of the third register (it means that the compiler must allocate 3 GPRs in all multipass vertex shader modes).

19.3.2 *Pixel shaders*

In the case of pixel shaders, if GEN_INDEX_PIX is set, the data will be put in the x field of the param_gen_pos+1 register.



The Auto Count Value is broadcast to all GPRs. It is loaded into a register wich has its LSBs hardwired to the GPR number (0 thru 63). Then if GEN_INDEX is high, the mux selects the auto-count value and it is loaded into the GPRs to be either used to retrieve data using the TP or sent to the SX for the RB to use it to write the data to memory

Figure 12: GPR input mux Control

20. State management

Every clock, the sequencer will report to the CP the oldest states still in the pipe. These are the states of the programs as they enter the last ALU clause.

20.1 Parameter cache synchronization

In order for the sequencer not to begin a group of pixels before the associated group of vertices has finished, the sequencer will keep a 6 bit count per state (for a total of 8 counters). These counters are initialized to 0 and every time a vertex shader exports its data TO THE PARAMETER CACHE, the corresponding pointer is incremented. When the SC sends a new vector of pixels with the SC_SQ_new_vector bit asserted, the sequencer will first check if the count is greater than 0 before accepting the transmission (it will in fact accept the transmission but then lower its ready to receive). Then the sequencer waits for the count to go to one and decrements it. The sequencer can then issue the group of pixels to the interpolators. Every time the state changes, the new state counter is initialized to 0.

21. XY Address imports

The SC will be able to send the XY addresses to the GPRs. It does so by interleaving the writes of the IJs (to the IJ buffer) with XY writes (to the XY buffer). Then when writing the data to the GPRs, the sequencer is going to interpolate the IJ data or pass the XY data thru a Fix→float converter and expander and write the converted values to the GPRs. The Xys are currently SCREEN SPACE COORDINATES. The values in the XY buffers will wrap. See section 19.2 for details on how to control the interpolation in this mode.

21.1 Vertex indexes imports

In order to import vertex indexes, we have 16 8x96 staging registers. These are loaded one line at a time by the VGT block (96 bits). They are loaded in floating point format and can be transferred in 4 or 8 clocks to the GPRs.

22. Registers

Please see the auto-generated web pages for register definitions.

23. Interfaces

23.1 External Interfaces

Whenever an x is used, it means that the bus is broadcast to all units of the same name. For example, if a bus is named SQ→SPx it means that SQ is going to broadcast the same information to all SP instances.

23.2 SC to SP Interfaces

23.2.1 *SC_SP#*

There is one of these interfaces at front of each of the SP (buffer to stage pixel interpolators). This interface transmits the I,J data for pixel interpolation. For the entire system, two quads per clock are transferred to the 4 SPs, so each of these 4 interfaces transmits one half of a quad per clock. The interface below describes a half of a quad worth of data.
The actual data which is transferred per quad is

Ref Pix I => S4.20 Floating Point I value *4
Ref Pix J => S4.20 Floating Point J value *4

This equates to a total of 200 bits which transferred over 2 clocks
and therefor needs an interface 100 bits wide

Additionally, X,Y data (12-bit unsigned fixed) is conditionally sent across this data bus over the same wires in an additional clock. The X,Y data is sent on the lower 24 bits of the data bus with faceness in the msb.
Transfers across these interfaces are synchronized with the SC_SQ IJ Control Bus transfers.

The data transfer across each of these busses is controlled by a IJ_BUF_INUSE_COUNT in the SC. Each time the SC has sent a pixel vector's worth of data to the SPs, he will increment the IJ_BUF_INUSE_COUNT count. Prior to sending the next pixel vectors data, he will check to make sure the count is less than MAX_BUFER_MINUS_2, if not the SC will stall until the SQ returns a pipelined pulse to decrement the count when he has scheduled a buffer free. Note: We could/may optimize for the case of only sending only IJ to use all the buffers to pre-load more. Currently it is planned for the SP to hold 2 double buffers of I,J data and two buffers of X,Y data, so if either X,Y or Centers and Centroids are on, then the SC can send two Buffers.

In at least the initial version, the SC shall send 16 quads per pixel vector even if the vector is not full. This will increment buffer write address pointers correctly all the time. (We may revisit this for both the SX,SP,SQ and add a EndOfVector signal on all interfaces to quit early. We opted for the simple mode first with a belief that only the end of packet and multiple new vector signals should cause a partial vector and that this would not really be significant performance hit.)

Name	**Bits**	**Description**
SC_SP#_data	100	IJ information sent over 2 clocks (or X,Y in 24 LSBs with faceness in upper bit) **Type 0 or 1**, First clock I, second clk J Field ULC URC LLC LRC Bits [63:39] [38:26] [25:13] [12:0] Format SE4M20 SE4M20 SE4M20 SE4M20 **Type 2** Field Face X Y Bits [24] [23:12] [11:0] Format Bit Unsigned Unsigned
SC_SP#_valid	1	Valid
SC_SP#_last_quad_data	1	This bit will be set on the last transfer of data per quad.
SC_SP#_type	2	0 -> Indicates centroids 1 -> Indicates centers 2 -> Indicates X,Y Data and faceness on data bus The SC shall look at state data to determine how many types to send for the

		interpolation process.

The # is included for clarity in the spec and will be replaced with a prefix of u#_ in the verilog module statement for the SC and the SP block will have neither because the instantiation will insert the prefix.

23.2.2 *SC_SQ*

This is the control information sent to the sequencer in order to synchronize and control the interpolation and/or loading data into the GPRs needed to execute a shader program on the sent pixels. This data will be sent over two clocks per transfer with 1 to 16 transfers. Therefore the bus (approx 108 bits) could be folded in half to approx 54 bits.

Name	**Bits**	**Description**
SC_SQ_data	46	Control Data sent to the SQ 1 clk transfers Event – valid data consist of event_id and state_id. Instruct SQ to post an event vector to send state id and event_id through request fifo and onto the reservation stations making sure state id and/or event_id gets back to the CP. Events only follow end of packets so no pixel vectors will be in progress. Empty Quad Mask – Transfer Control data consisting of pc_dealloc or new_vector. Receipt of this is to transfer pc_dealloc or new_vector without any valid quad data. New vector will always be posted to request fifo and pc_dealloc will be attached to any pixel vector outstanding or posted in request fifo if no valid quad outstanding. 2 clk transfers Quad Data Valid – Sending quad data with or without new_vector or pc_dealloc. New vector will be posted to request fifo with or without a pixel vector and pc_dealloc will be posted with a pixel vector unless none is in progress. In this case the pc_dealloc will be posted in the request queue. Filler quads will be transferred with The Quad mask set but the pixel corresponding pixel mask set to zero.
SC_SQ_valid	1	SC sending valid data, 2nd clk could be all zeroes

SC_SQ_data – first clock and second clock transfers are shown in the table below.

Name	**BitField**	**Bits**	**Description**
1st Clock Transfer			
SC_SQ_event	0	1	This transfer is a 1 clock event vector Force quad_mask = new_vector=pc_dealloc=0
SC_SQ_event_id	[45:1]	4	This field identifies the event 0 => denotes an End Of State Event 1

			=> TBD
SC_SQ_state_id	[~~7~~8:6~~5~~]	3	State/constant pointer (6*3+3)
SC_SQ_pc_dealloc	[~~10~~11:~~8~~9]	3	Deallocation token for the Parameter Cache
SC_SQ_new_vector	~~11~~12	1	The SQ must wait for Vertex shader done count > 0 and after dispatching the Pixel Vector the SQ will decrement the count.
SC_SQ_quad_mask	[~~15~~16:~~12~~13]	4	Quad Write mask left to right SP0 => SP3
SC_SQ_end_of_prim	~~16~~17	1	End Of the primitive
SC_SQ_pix_mask	[~~32~~33:~~17~~18]	16	Valid bits for all pixels SP0=>SP3 (UL,UR,LL,LR)
SC_SQ_provok_vtx	[~~34~~35:~~33~~34]	2	Provoking vertex for flat shading
SC_SQ_lod_correct_0	[~~43~~44:~~35~~36]	9	LOD correction for quad 0 (SP0) (9 bits per quad)
SC_SQ_lod_correct_1	[~~52~~53:~~44~~45]	9	LOD correction for quad 1 (SP1) (9 bits per quad)
2nd Clock Transfer			
SC_SQ_lod_correct_2	[8:0]	9	LOD correction for quad 2 (SP2) (9 bits per quad)
SC_SQ_lod_correct_3	[17:9]	9	LOD correction for quad 3 (SP3) (9 bits per quad)
SC_SQ_pc_ptr0	[28:18]	11	Parameter Cache pointer for vertex 0
SC_SQ_pc_ptr1	[39:29]	11	Parameter Cache pointer for vertex 1
SC_SQ_pc_ptr2	[50:40]	11	Parameter Cache pointer for vertex 2
SC_SQ_prim_type	[53:51]	3	Stippled line and Real time command need to load tex cords from alternate buffer 000: Sprite (point) 001: Line 010: Tri_rect 100: Realtime Sprite (point) 101: Realtime Line 110: Realtime Tri_rect

Name	Bits	Description
SQ_SC_free_buff	1	Pipelined bit that instructs SC to decrement count of buffers in use.
SQ_SC_dec_cntr_cnt	1	Pipelined bit that instructs SC to decrement count of new vector and/or event sent to prevent SC from overflowing SQ interpolator/Reservation request fifo.

The scan converter will submit a partial vector whenever:

1.) He gets a primitive marked with an end of packet signal.

2.) A current pixel vector is being assembled with at least one or more valid quads and the vector has been marked for deallocate when a primitive marked new_vector arrives. The Scan Converter will submit a partial vector (up to 16quads with zero pixel mask to fill out the vector) prior to submitting the new_vector marker\primitive.

(This will prevent a hang which can be demonstrated when all primitives in a packet three vectors are culled except for a one quad primitive that gets marked pc_dealloc (vertices maximum size). In this case two new_vectors are submitted and processed, but then one valid quad with the pc_dealloc creates a vector and then the new would wait for another vertex vector to be processed, but the one being waited for could never export until the pc_dealloc signal made it through and thus the hang.)

23.2.3 *SQ to SX(SP): Interpolator bus*

Name	Direction	Bits	Description
SQ_SPx_interp_flat_vtx	SQ→SPx	2	Provoking vertex for flat shading
SQ_SPx_interp_flat_gouraud	SQ→SPx	1	Flat or gouraud shading
SQ_SPx_interp_cyl_wrap	SQ→SPx	4	Wich channel needs to be cylindrical wrapped
SQ_SPx_interp_param_gen	SQ→SPx	1	Generate Parameter
SQ_SPx_interp_prim_type	SQ→SPx	2	Bits [1:0] of primitive type sent by SC
SQ_SPx_interp_buff_swap	SQ→SPx	1	Swapp IJ buffers
SQ_SPx_interp_IJ_line	SQ→SPx	2	IJ line number
SQ_SPx_interp_mode	SQ→SPx	1	Center/Centroid sampling
SQ_SXx_pc_ptr0	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_pc_ptr1	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_pc_ptr2	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_rt_sel	SQ→SXx	1	Selects between RT and Normal data (Bit 2 of prim type)
SQ_SX0_pc_wr_en	SQ→SX0	8	Write enable for the PC memories
SQ_SX1_pc_wr_en	SQ→SX1	8	Write enable for the PC memories
SQ_SXx_pc_wr_addr	SQ→SXx	7	Write address for the PCs
SQ_SXx_pc_channel_mask	SQ→SXx	4	Channel mask
SQ_SXx_pc_ptr_valid	SQ→SXx	1	Read pointers are valid.
SQ_SPx_interp_valid	SQ→SPx	1	Interpolation control valid

23.2.4 *SQ to SP: Staging Register Data*

This is a broadcast bus that sends the VSISR information to the staging registers of the shader pipes.

Name	Direction	Bits	Description
SQ_SPx_vsr_data	SQ→SPx	96	Pointers of indexes or HOS surface information
SQ_SPx_vsr_double	SQ→SPx	1	0: Normal 96 bits per vert 1: double 192 bits per vert
SQ_SP0_ vsr_valid	SQ→SP0	1	Data is valid
SQ_SP1_ vsr_ valid	SQ→SP1	1	Data is valid
SQ_SP2_ vsr_ valid	SQ→SP2	1	Data is valid
SQ_SP3_ vsr_ valid	SQ→SP3	1	Data is valid
SQ_SPx_vsr_read	SQ→SPx	1	Increment the read pointers

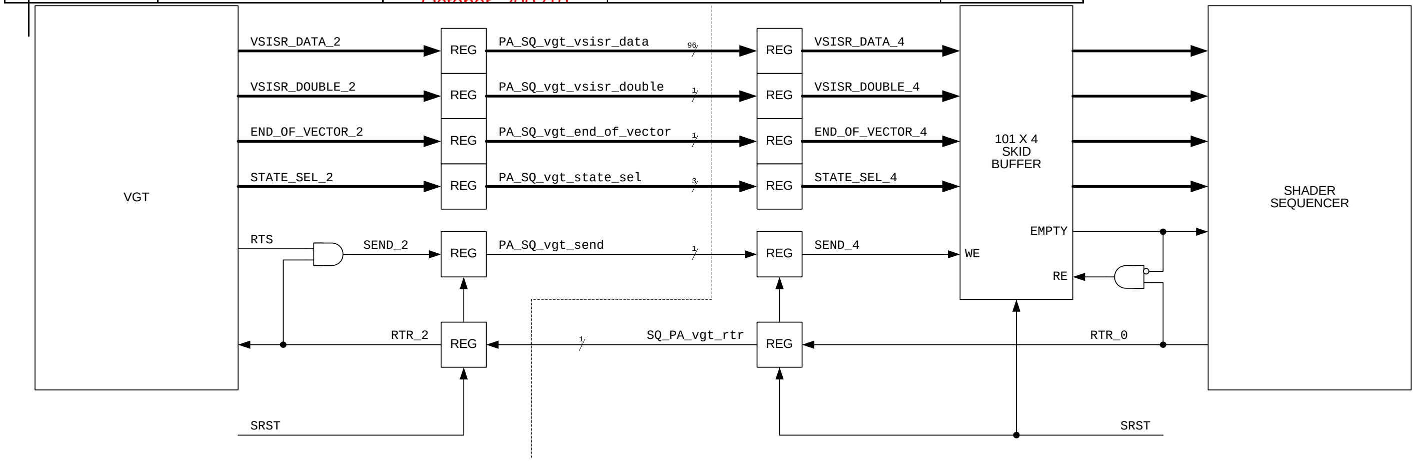
23.2.5 *VGT to SQ : Vertex interface*

23.2.5.1 Interface Signal Table

The area difference between the two methods is not sufficient to warrant complicating the interface or the state requirements of the VSISRs. **Therefore, the POR for this interface is that the VGT will transmit the data to the VSISRs (via the Shader Sequencer) in full, 32-bit floating-point format.** The VGT can transmit up to six 32-bit floating-point values to each VSISR where four or more values require two transmission clocks. The data bus is 96 bits wide. In the case where an event is sent the 5 LSBs of VGT_SQ_vsisr_data contain the eventID.

Name	Bits	Description
VGT_SQ_vsisr_data	96	Pointers of indexes or HOS surface information
VGT_SQ_event	1	VGT is sending an event
VGT_SQ_vsisr_continued	1	0: Normal 96 bits per vert 1: double 192 bits per vert
VGT_SQ_end_of_vtx_vect	1	Indicates the last VSISR data set for the current process vector (for double vector data, "end_of_vector" is set on the first vector)
VGT_SQ_indx_valid	1	Vsisr data is valid
VGT_SQ_state	3	Render State (6*3+3 for constants). This signal is guaranteed to be correct when “VGT_SQ_vgt_end_of_vector” is high.
VGT_SQ_send	1	Data on the VGT_SQ is valid receive (see write-up for standard R400 SEND/RTR interface handshaking)
SQ_VGT_rtr	1	Ready to receive (see write-up for standard R400 SEND/RTR interface handshaking)

23.2.5.2 Interface Diagrams


VGT
VSISR_DATA_2
VSISR_DOUBLE_2
END_OF_VECTOR_2
STATE_SEL_2
RTS
SEND_2
RTR_2
SRST
REG
PA_SQ_vgt_vsisr_data
PA_SQ_vgt_vsisr_double
PA_SQ_vgt_end_of_vector
PA_SQ_vgt_state_sel
PA_SQ_vgt_send
SQ_PA_vgt_rtr
96
1
1
3
1
1
VSISR_DATA_4
VSISR_DOUBLE_4
END_OF_VECTOR_4
STATE_SEL_4
SEND_4
101 X 4
SKID
BUFFER
EMPTY
WE
RE
RTR_0
SRST
SHADER
SEQUENCER

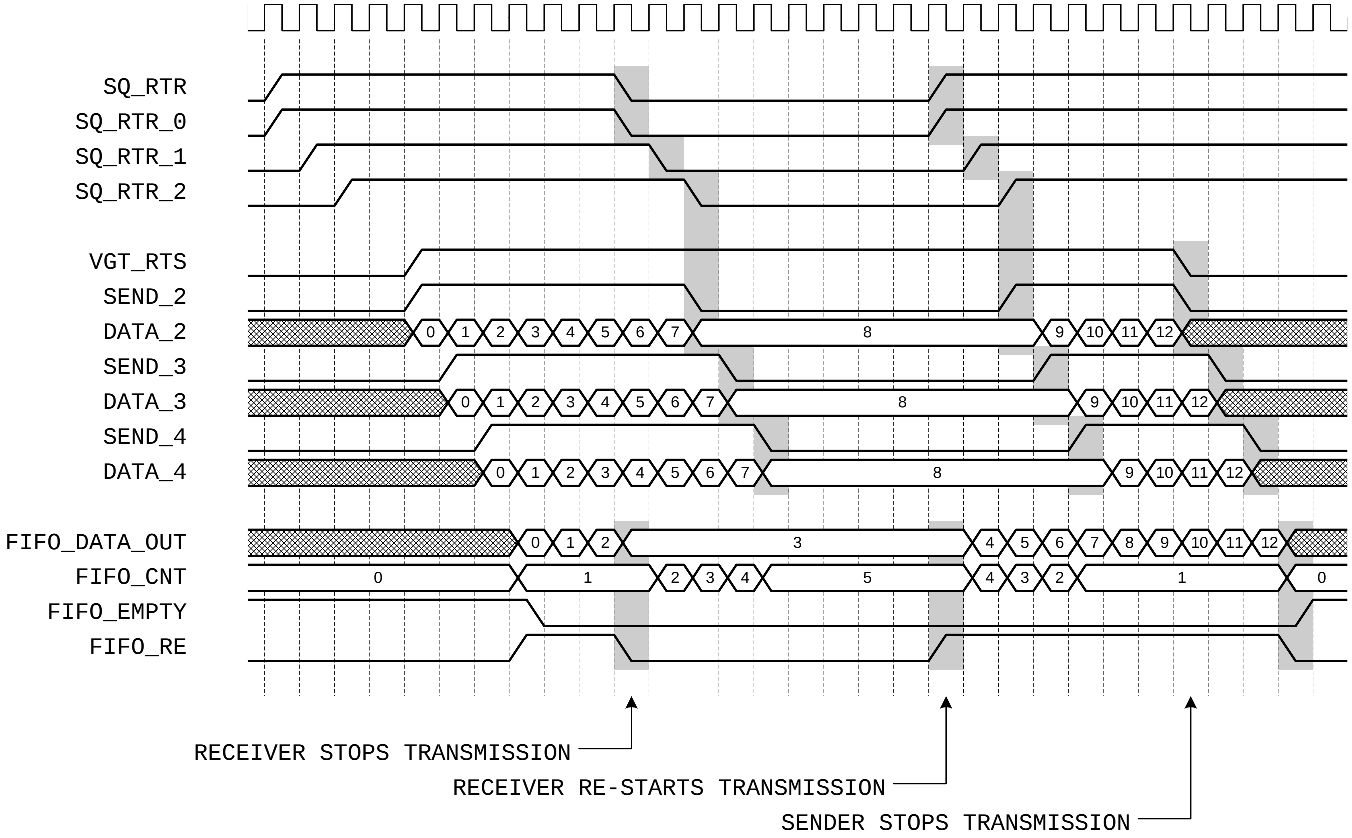


Figure 1. Detailed Logical Diagram for PA_SQ_vgt Interface.

23.2.6 *SQ to SX: Control bus*

Name	Direction	Bits	Description
SQ_SXx_exp_type	SQ→SXx	2	00: Pixel without z (1 to 4 buffers) 01: Pixel with z (1 to 4 buffers) 10: Position (1 or 2 results) 11: Pass thru (4,8 or 12 results aligned)
SQ_SXx_exp_number	SQ→SXx	2	Number of locations needed in the export buffer (encoding depends on the type see bellow).
SQ_SXx_exp_alu_id	SQ→SXx	1	ALU ID
SQ_SXx_exp_valid	SQ→SXx	1	Valid bit
SQ_SXx_exp_state	SQ→SXx	3	State Context
SQ_SXx_free_done	SQ→SXx	1	Pulse that indicates that the previous export is finished **from the point of view of the SP. This does not necessarily mean that the data has been transferred to RB or PA, or that the space in export buffer for that particular vector thread has been freed up.**
SQ_SXx_free_alu_id	SQ→SXx	1	ALU ID

Depending on the type the number of export location changes:

- Type 00 : Pixels without Z
  - 00 = 1 buffer
  - 01 = 2 buffers
  - 10 = 3 buffers
  - 11 = 4 buffer
- Type 01: Pixels with Z
  - 00 = 2 Buffers (color + Z)
  - 01 = 3 buffers (2 color + Z)
  - 10 = 4 buffers (3 color + Z)
  - 11 = 5 buffers (4 color + Z)
- Type 10 : Position export
  - 00 = 1 position
  - 01 = 2 positions
  - 1X = Undefined
- Type 11: Pass Thru
  - 00 = 4 buffers
  - 01 = 8 buffers
  - 10 = 12 buffers
  - 11 = Undefined

Below the thick black line is the end of transfer packet that tells the SX that a given export is finished. The report packet **will always arrive either before or at the same time than the next export to the same ALU id**.

23.2.7 *SX to SQ : Output file control*

Name	Direction	Bits	Description
SXx_SQ_exp_count_rdy	SXx→SQ	1	Raised by SX0 to indicate that the following two fields reflect the result of the most recent export
SXx_SQ_exp_pos_avail	SXx→SQ	2	Specifies whether there is room for another position. 00 : 0 buffers ready 01 : 1 buffer ready 10 : 2 or more buffers ready
SXx_SQ_exp_buf_avail	SXx→SQ	7	Specifies the space available in the output buffers. 0: buffers are full 1: 2K-bits available (32-bits for each of the 64

			pixels in a clause) ... 64: 128K-bits available (16 128-bit entries for each of 64 pixels) 65-127: RESERVED

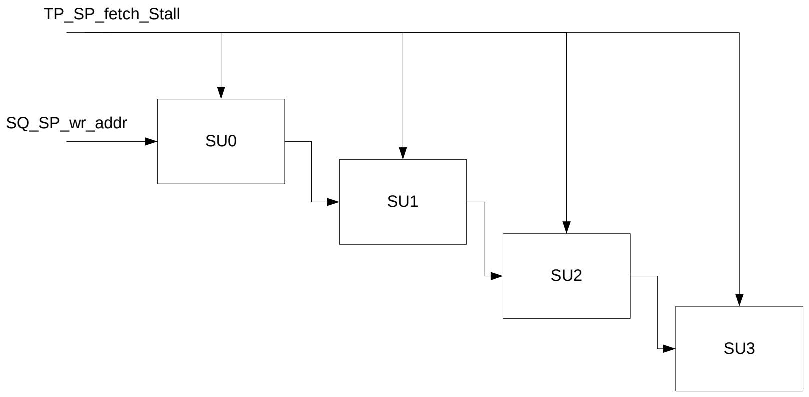
23.2.8 *SQ to TP: Control bus*

Once every clock, the fetch unit sends to the sequencer on which RS line it is now working and if the data in the GPRs is ready or not. This way the sequencer can update the fetch valid bits flags for the reservation station. The sequencer also provides the instruction and constants for the fetch to execute and the address in the register file where to write the fetch return data.

Name	Direction	Bits	Description
TPx_SQ_data_rdy	TPx→ SQ	1	Data ready
TPx_SQ_rs_line_num	TPx→ SQ	6	Line number in the Reservation station
TPx_SQ_type	TPx→ SQ	1	Type of data sent (0:PIXEL, 1:VERTEX)
SQ_TPx_send	SQ→TPx	1	Sending valid data
SQ_TPx_const	SQ→TPx	48	Fetch state sent over 4 clocks (192 bits total)
SQ_TPx_instr	SQ→TPx	24	Fetch instruction sent over 4 clocks
SQ_TPx_end_of_group	SQ→TPx	1	Last instruction of the group
SQ_TPx_Type	SQ→TPx	1	Type of data sent (0:PIXEL, 1:VERTEX)
SQ_TPx_gpr_phase	SQ→TPx	2	Write phase signal
SQ_TP0_lod_correct	SQ→TP0	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP0_pix_mask	SQ→TP0	4	Pixel mask 1 bit per pixel
SQ_TP1_lod_correct	SQ→TP1	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP1_pix_mask	SQ→TP1	4	Pixel mask 1 bit per pixel
SQ_TP2_lod_correct	SQ→TP2	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP2_pix_mask	SQ→TP2	4	Pixel mask 1 bit per pixel
SQ_TP3_lod_correct	SQ→TP3	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP3_pix_mask	SQ→TP3	4	Pixel mask 1 bit per pixel
SQ_TPx_rs_line_num	SQ→TPx	6	Line number in the Reservation station
SQ_TPx_write_gpr_index	SQ->TPx	7	Index into Register file for write of returned Fetch Data
SQ_TPx_ctx_id	SQ→TPx	3	The state context ID (needed for multisample resolves)

23.2.9 *TP to SQ: Texture stall*

The TP sends this signal to the SQ and the SPs when its input buffer is full.



Name	Direction	Bits	Description
TP_SQ_fetch_stall	TP→ SQ	1	Do not send more texture request if asserted

23.2.10 *SQ to SP: Texture stall*

Name	Direction	Bits	Description
SQ_SPx_fetch_stall	SQ→SPx	1	Do not send more texture request if asserted

23.2.11 *SQ to SP: GPR and auto counter*

Name	Direction	Bits	Description
SQ_SPx_gpr_wr_addr	SQ→SPx	7	Write address
SQ_SPx_gpr_rd_addr	SQ→SPx	7	Read address
SQ_SPx_gpr_rd_en	SQ→SPx	1	Read Enable
SQ_SP0_gpr_wr_en	SQ→SPx	4	Write Enable for the GPRs of SP0
SQ_SP1_gpr_wr_en	SQ→SPx	4	Write Enable for the GPRs of SP1
SQ_SP2_gpr_wr_en	SQ→SPx	4	Write Enable for the GPRs of SP2
SQ_SP3_gpr_wr_en	SQ→SPx	4	Write Enable for the GPRs of SP3
SQ_SPx_gpr_phase	SQ→SPx	2	The phase mux (arbitrates between inputs, ALU SRC reads and writes)
SQ_SPx_channel_mask	SQ→SPx	4	The channel mask
SQ_SPx_gpr_input_sel	SQ→SPx	2	When the phase mux selects the inputs this tells from which source to read from: Interpolated data, VTX0, VTX1, autogen counter.
SQ_SPx_auto_count	SQ→SPx	21	Auto count generated by the SQ, common for all shader pipes

23.2.12 *SQ to SPx: Instructions*

Name	Direction	Bits	Description
SQ_SPx_instr_start	SQ→SPx	1	Instruction start
SQ_SP_instr	SQ→SPx	24	Transferred over 4 cycles 0: SRC A Negate Argument Modifier 0:0 SRC A Abs Argument Modifier 1:1 SRC A Swizzle 9:2 Vector Dst 15:10 Per channel Select 23:16 00: GPR 01: PV 10: PS 11: Constant (if 11 has to be 11 for all channels) -- - 1: SRC B Negate Argument Modifier 0:0 SRC B Abs Argument Modifier 1:1 SRC B Swizzle 9:2 Scalar Dst 15:10 Per channel Select 23:16 00: GPR 01: PV 10: PS 11: Constant (if 11 has to be 11 for all channels) -- - 2: SRC C Negate Argument Modifier 0:0 SRC C Abs Argument Modifier 1:1 SRC C Swizzle 9:2 Unused 15:10 Per channel Select 23:16 00: GPR 01: PV 10: PS 11: Constant (if 11 has to be 11 for all channels) -- - 3: Vector Opcode 4:0 Scalar Opcode 10:5 Vector Clamp 11:11 Scalar Clamp 12:12 Vector Write Mask 16:13 Scalar Write Mask 20:17 Unused 23:21
SQ_SP0_pred_override	SQ→SP0	4	0: Use per channel RGBA field (enables the per channel logic, if not set only pay attention to the 11 seting). 1: Use GPR
SQ_SP1_pred_override	SQ→SP1	4	0: Use per channel RGBA field (enables the per channel logic, if not set only pay attention to the 11 seting). 1: Use GPR
SQ_SP2_pred_override	SQ→SP2	4	0: Use per channel RGBA field (enables the per channel logic, if not set only pay attention to the 11 seting). 1: Use GPR
SQ_SP3_pred_override	SQ→SP3	4	0: Use per channel RGBA field (enables the per channel logic, if not set only pay attention to the 11

			seting). 1: Use GPR
SQ_SPx_exp_id	SQ→SPx	1	GPR ID
SQ_SPx_exporting	SQ→SPx	1	0: Not Exporting 1: Exporting
SQ_SPx_stall	SQ→SPx	1	Stall signal

23.2.13 *SQ to SX: write mask interface (must be aligned with the SP data)*

Name	Direction	Bits	Description
SQ_SX0_write_mask	SQ→SP0	8	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock. This is for the data coming of SP0 and SP2.
SQ_SX1_ write_mask	SQ→SP1	8	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock. This is for the data coming of SP1 and SP3.

23.2.14 *SP to SQ: Constant address load/ Predicate Set/Kill set*

Name	Direction	Bits	Description
SP0_SQ_const_addr	SP0→SQ	36	Constant address load / predicate vector load (4 bits only)/ Kill vector load (4 bits only) to the sequencer
SP0_SQ_valid	SP0→SQ	1	Data valid
SP1_SQ_const_addr	SP1→SQ	36	Constant address load / predicate vector load (4 bits only)/ Kill vector load (4 bits only) to the sequencer
SP1_SQ_valid	SP1→SQ	1	Data valid
SP2_SQ_const_addr	SP2→SQ	36	Constant address load / predicate vector load (4 bits only)/ Kill vector load (4 bits only) to the sequencer
SP2_SQ_valid	SP2→SQ	1	Data valid
SP3_SQ_const_addr	SP3→SQ	36	Constant address load / predicate vector load (4 bits only)/ Kill vector load (4 bits only) to the sequencer
SP3_SQ_valid	SP3→SQ	1	Data valid
SP0_SQ_data_type	SP→SQ	2	Data Type 0: Constant Load 1: Predicate Set 2: Kill vector load

Because of the sharing of the bus none of the MOVA, PREDSET or KILL instructions may be coissued.

23.2.15 *SQ to SPx: constant broadcast*

Name	Direction	Bits	Description
SQ_SPx_const	SQ→SPx	128	Constant broadcast

23.2.16 *SQ to CP: RBBM bus*

Name	Direction	Bits	Description
SQ_RBB_rs	SQ→CP	1	Read Strobe
SQ_RBB_rd	SQ→CP	32	Read Data
SQ_RBBM_nrtrtr	SQ→CP	1	Optional
SQ_RBBM_rtr	SQ→CP	1	Real-Time (Optional)

23.2.17 *CP to SQ: RBBM bus*

Name	Direction	Bits	Description
rbbm_we	CP→SQ	1	Write Enable
rbbm_a	CP→SQ	15	Address -- Upper Extent is TBD (16:2)
rbbm_wd	CP→SQ	32	Data

rbbm_be	CP→SQ	4	Byte Enables
rbbm_re	CP→SQ	1	Read Enable
rbb_rs0	CP→SQ	1	Read Return Strobe 0
rbb_rs1	CP→SQ	1	Read Return Strobe 1
rbb_rd0	CP→SQ	32	Read Data 0
rbb_rd1	CP→SQ	32	Read Data 0
RBBM_SQ_soft_reset	CP→SQ	1	Soft Reset

23.2.18 *SQ to CP: State report*

Name	Direction	Bits	Description
SQ_CP_vs_event	SQ→CP	1	Vertex Shader Event
SQ_CP_vs_eventid	SQ→CP	~~4~~5	Vertex Shader Event ID
SQ_CP_ps_event	SQ→CP	1	Pixel Shader Event
SQ_CP_ps_eventid	SQ→CP	~~4~~5	Pixel Shader Event ID

23.3 Example of control flow program execution

We now provide some examples of execution to better illustrate the new design.

Given the program:

Alu 0
Alu 1
Tex 0
Tex 1
Alu 3 Serial
Alu 4
Tex 2
Alu 5
Alu 6 Serial
Tex 3
Alu 7
Alloc Position 1 buffer
Alu 8 Export
Tex 4
Alloc Parameter 3 buffers
Alu 9 Export 0
Tex 5
Alu 10 Serial Export 2
Alu 11 Export 1 End

Would be converted into the following CF instructions:

```
Execute 0 Alu 0 Alu 0 Tex 0 Tex 1 Alu 0 Alu 0 Tex 0 Alu 1 Alu 0 Tex
Execute 0 Alu
Alloc Position 1
Execute 0 Alu 0 Tex
Alloc Param 3
Execute_end 0 Alu 0 Tex 1 Alu 0 Alu
```

And the execution of this program would look like this:

Put thread in Vertex RS:

Control Flow Instruction Pointer (12 bits), (CFP)
Execution Count Marker (3 or 4 bits), (ECM)
Loop Iterators (4x9 bits), (LI)
Call return pointers (4x12 bits), (CRP)

Predicate Bits(4x64 bits), (PB)
Export ID (1 bit), (EXID)
GPR Base Ptr (8 bits), (GPR)
Export Base Ptr (7 bits), (EB)
Context Ptr (3 bits).(CPTR)
LOD correction bits (16x6 bits) (LOD)

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	0	0	0	0	0	0	0	0	0

Valid Thread (VALID)
Texture/ALU engine needed (TYPE)
Texture Reads are outstanding (PENDING)
Waiting on Texture Read to Complete (SERIAL)
Allocation Wait (2 bits) (ALLOC)
    00 – No allocation needed
    01 – Position export allocation needed (ordered export)
    10 – Parameter or pixel export needed (ordered export)
    11 – pass thru (out of order export)
Allocation Size (4 bits) (SIZE)
Position Allocated (POS_ALLOC)
First thread of a new context (FIRST)
Last (1 bit), (LAST)

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	0	0	0	0	0	1	0

Then the thread is picked up for the execution of the first control flow instruction:

```
Execute 0 Alu 0 Alu 0 Tex 0 Tex 1 Alu 0 Alu 0 Tex 0 Alu 1 Alu 0 Tex
```

It executes the first two ALU instructions and goes back to the RS for a resource request change. Here is the state returned to the RS:

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	2	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	0	0	0	0	0	1	0

Then when the texture pipe frees up, the arbiter picks up the thread to issue the texture reads. The thread comes back in this state:

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	4	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	1	0	0	0	1	0

Because of the serial bit the arbiter must wait for the texture to return and clear the PENDING bit before it can pick the thread up. Lets say that the texture reads are complete, then the arbiter picks up the thread and returns it in this state:

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	6	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	0	0	0	0	0	1	0

Again the TP frees up, the arbiter picks up the thread and executes. It returns in this state:

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	7	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	0	0	0	1	0

Now, even if the texture has not returned we can still pick up the thread for ALU execution because the serial bit is not set. The thread will however come back to the RS for the second ALU instruction because it has the serial bit set.

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	8	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	1	0	0	0	1	0

As soon as the TP clears the pending bit the thread is picked up and returns:

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	9	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	0	0	0	0	0	1	0

Picked up by the TP and returns:

```
Execute 0 Alu
```

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
1	0	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	0	0	0	1	0

Picked up by the ALU and returns (lets say the TP has not returned yet):

```
Alloc Position 1
```

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
2	0	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	01	1	0	1	0

If the SX has the place for the export, the SQ is going to allocate and pick up the thread for execution. It returns to the RS in this state:

```
Execute 0 Alu 0 Tex
```

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
3	1	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	1	0	0	0	1	1	0

Now, since the TP has not returned yet, we must wait for it to return because we cannot issue multiple texture requests. The TP returns, clears the PENDING bit and we proceed:

```
Alloc Param 3
```

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
4	0	0	0	0	1	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	10	3	1	1	0

Once again the SQ makes sure the SX has enough room in the Parameter cache before it can pick up this thread.

```
Execute_end 0 Alu 0 Tex 1 Alu 0 Alu
```

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
5	1	0	0	0	1	0	100	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	1	0	0	0	1	1	0

This executes on the TP and then returns:

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
5	2	0	0	0	1	0	100	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	1	0	0	1	1	1

Waits for the TP to return because of the textures reads are pending (and SERIAL in this case). Then executes and does not return to the RS because the LAST bit is set. This is the end of this thread and before dropping it on the floor, the SQ notifies the SX of export completion.

24. Open issues

Need to do some testing on the size of the register file as well as on the register file allocation method (dynamic VS static).

Saving power?

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 2015~~1~~4 October, 200211	GEN-CXXXXX-REVA	1 of 51

Author: Laurent Lefebvre

Issue To:	**Copy No:**

R400 Sequencer Specification

SQ

Version 2.07

Overview: This is an architectural specification for the R400 Sequencer block (SEQ). It provides an overview of the required capabilities and expected uses of the block. It also describes the block interfaces, internal sub-blocks, and provides internal state diagrams.

AUTOMATICALLY UPDATED FIELDS:

Document Location:	C:\perforce\r400\doc_lib\design\blocks\sq\R400_Sequencer.doc
Current Intranet Search Title:	R400 Sequencer Specification

APPROVALS	
Name/Dept	Signature/Date

Remarks:

Table Of Contents

1. OVERVIEW ... 7
1.1 Top Level Block Diagram ... 9
1.2 Data Flow graph (SP) ... 10
1.3 Control Graph ... 11
2. INTERPOLATED DATA BUS ... 11
3. INSTRUCTION STORE ... 14
4. SEQUENCER INSTRUCTIONS ... 14
5. CONSTANT STORES ... 14
5.1 Memory organizations ... 14
5.2 Management of the Control Flow Constants ... 15
5.3 Management of the re-mapping tables ... 15
5.3.1 R400 Constant management ... 15
5.3.2 Proposal for R400LE constant management ... 15
5.3.3 Dirty bits ... 17
5.3.4 Free List Block ... 17
5.3.5 De-allocate Block ... 18
5.3.6 Operation of Incremental model ... 18
5.4 Constant Store Indexing ... 18
5.5 Real Time Commands ... 19
5.6 Constant Waterfalling ... 19
6. LOOPING AND BRANCHES ... 20
6.1 The controlling state. ... 20
6.2 The Control Flow Program ... 20
6.2.1 Control flow instructions table ... 21
6.3 Implementation ... 23
6.4 Data dependant predicate instructions ... 24
6.5 HW Detection of PV,PS ... 25
6.6 Register file indexing ... 25
6.7 Debugging the Shaders ... 26~~25~~
6.7.1 Method 1: Debugging registers ... 26
6.7.2 Method 2: Exporting the values in the GPRs ... 26
7. PIXEL KILL MASK ... 26
8. MULTIPASS VERTEX SHADERS (HOS) ... 26
9. REGISTER FILE ALLOCATION ... 27~~26~~
10. FETCH ARBITRATION ... 28
11. ALU ARBITRATION ... 28
12. HANDLING STALLS ... 29
13. CONTENT OF THE RESERVATION STATION FIFOS ... 29
14. THE OUTPUT FILE ... 29
15. IJ FORMAT ... 29
15.1 Interpolation of constant attributes ... 29
16. STAGING REGISTERS ... 30

17. THE PARAMETER CACHE 31
17.1 Export restrictions 32
17.1.1 Pixel exports: 32
17.1.2 Vertex exports: 32
17.1.3 Pass thru exports: 32
17.2 Arbitration restrictions 32
18. EXPORT TYPES 32
18.1 Vertex Shading 32
18.2 Pixel Shading 33
19. SPECIAL INTERPOLATION MODES 33
19.1 Real time commands 33
19.2 Sprites/ XY screen coordinates/ FB information 33
19.3 Auto generated counters 34
19.3.1 Vertex shaders 34
19.3.2 Pixel shaders 34
20. STATE MANAGEMENT 35
20.1 Parameter cache synchronization 35
21. XY ADDRESS IMPORTS 35
21.1 Vertex indexes imports 35
22. REGISTERS 35
23. INTERFACES 36
23.1 External Interfaces 36
23.2 SC to SP Interfaces 36
23.2.1 SC_SP# 36
23.2.2 SC_SQ 37
23.2.3 SQ to SX(SP): Interpolator bus 39
23.2.4 SQ to SP: Staging Register Data 39
23.2.5 VGT to SQ : Vertex interface 39
23.2.6 SQ to SX: Control bus 42
23.2.7 SX to SQ : Output file control 42
23.2.8 SQ to TP: Control bus 43
23.2.9 TP to SQ: Texture stall 43
23.2.10 SQ to SP: Texture stall 44
23.2.11 SQ to SP: GPR and auto counter 44
23.2.12 SQ to SPx: Instructions 45
23.2.13 SP to SQ: Constant address load/ Predicate Set/Kill set 46
23.2.14 SQ to SPx: constant broadcast 46
23.2.15 SQ to CP: RBBM bus 46
23.2.16 CP to SQ: RBBM bus 46
23.2.17 SQ to CP: State report 47
23.3 Example of control flow program execution 47

24. OPEN ISSUES.. 51

Revision Changes:

Rev 0.1 (Laurent Lefebvre) Date: May 7, 2001	First draft.
Rev 0.2 (Laurent Lefebvre) Date : July 9, 2001	Changed the interfaces to reflect the changes in the SP. Added some details in the arbitration section.
Rev 0.3 (Laurent Lefebvre) Date : August 6, 2001	Reviewed the Sequencer spec after the meeting on August 3, 2001.
Rev 0.4 (Laurent Lefebvre) Date : August 24, 2001	Added the dynamic allocation method for register file and an example (written in part by Vic) of the flow of pixels/vertices in the sequencer.
Rev 0.5 (Laurent Lefebvre) Date : September 7, 2001	Added timing diagrams (Vic)
Rev 0.6 (Laurent Lefebvre) Date : September 24, 2001	Changed the spec to reflect the new R400 architecture. Added interfaces.
Rev 0.7 (Laurent Lefebvre) Date : October 5, 2001	Added constant store management, instruction store management, control flow management and data dependant predication.
Rev 0.8 (Laurent Lefebvre) Date : October 8, 2001	Changed the control flow method to be more flexible. Also updated the external interfaces.
Rev 0.9 (Laurent Lefebvre) Date : October 17, 2001	Incorporated changes made in the 10/18/01 control flow meeting. Added a NOP instruction, removed the conditional_execute_or_jump. Added debug registers.
Rev 1.0 (Laurent Lefebvre) Date : October 19, 2001	Refined interfaces to RB. Added state registers.
Rev 1.1 (Laurent Lefebvre) Date : October 26, 2001	Added SEQ→SP0 interfaces. Changed delta precision. Changed VGT→SP0 interface. Debug Methods added.
Rev 1.2 (Laurent Lefebvre) Date : November 16, 2001	Interfaces greatly refined. Cleaned up the spec.
Rev 1.3 (Laurent Lefebvre) Date : November 26, 2001	Added the different interpolation modes.
Rev 1.4 (Laurent Lefebvre) Date : December 6, 2001	Added the auto incrementing counters. Changed the VGT→SQ interface. Added content on constant management. Updated GPRs.
Rev 1.5 (Laurent Lefebvre) Date : December 11, 2001	Removed from the spec all interfaces that weren't directly tied to the SQ. Added explanations on constant management. Added PA→SQ synchronization fields and explanation.
Rev 1.6 (Laurent Lefebvre) Date : January 7, 2002	Added more details on the staging register. Added detail about the parameter caches. Changed the call instruction to a Conditionnal_call instruction. Added details on constant management and updated the diagram.
Rev 1.7 (Laurent Lefebvre) Date : February 4, 2002	Added Real Time parameter control in the SX interface. Updated the control flow section.
Rev 1.8 (Laurent Lefebvre) Date : March 4, 2002	New interfaces to the SX block. Added the end of clause modifier, removed the end of clause instructions.
Rev 1.9 (Laurent Lefebvre) Date : March 18, 2002	Rearangement of the CF instruction bits in order to ensure byte alignement.
Rev 1.10 (Laurent Lefebvre) Date : March 25, 2002	Updated the interfaces and added a section on exporting rules.
Rev 1.11 (Laurent Lefebvre) Date : April 19, 2002	Added CP state report interface. Last version of the spec with the old control flow scheme
Rev 2.0 (Laurent Lefebvre) Date : April 19, 2002	New control flow scheme

Rev 2.01 (Laurent Lefebvre) Date : May 2, 2002	Changed slightly the control flow instructions to allow force jumps and calls.
Rev 2.02 (Laurent Lefebvre) Date : May 13, 2002	Updated the Opcodes. Added type field to the constant/pred interface. Added Last field to the SQ→SP instruction load interface.
Rev 2.03 (Laurent Lefebvre) Date : July 15, 2002	SP interface updated to include predication optimizations. Added the predicate no stall instructions,
Rev 2.04 (Laurent Lefebvre) Date :August 2, 2002	Documented the new parameter generation scheme for XY coordinates points and lines STs.
Rev 2.05 (Laurent Lefebvre) Date : September 10, 2002	Some interface changes and an architectural change to the auto-counter scheme.
Rev 2.06 (Laurent Lefebvre) Date : October 11, 2002	Widened the event interface to 5 bits. Some other little typos corrected.
Rev 2.07 (Laurent Lefebvre) Date : October 14, 2002	Loops, jumps and calls are now using a 13 bit address which allows to jump and call and loop around any control flow addresses (does not requires to be even anymore).

1. Overview

The sequencer chooses two ALU threads and a fetch hread to execute, and executes all of the instructions in a block before looking for a new clause of the same type. Two ALU threads are executed interleaved to hide the ALU latency. The arbitrator will give priority to older threads. There are two separate reservation stations, one for pixel vectors and one for vertices vectors. This way a pixel can pass a vertex and a vertex can pass a pixel.

To support the shader pipe the sequencer also contains the shader instruction cache, constant store, control flow constants and texture state. The four shader pipes also execute the same instruction thus there is only one sequencer for the whole chip.

The sequencer first arbitrates between vectors of 64 vertices that arrive directly from primitive assembly and vectors of 16 quads (64 pixels) that are generated in the scan converter.

The vertex or pixel program specifies how many GPRs it needs to execute. The sequencer will not start the next vector until the needed space is available in the GPRs.

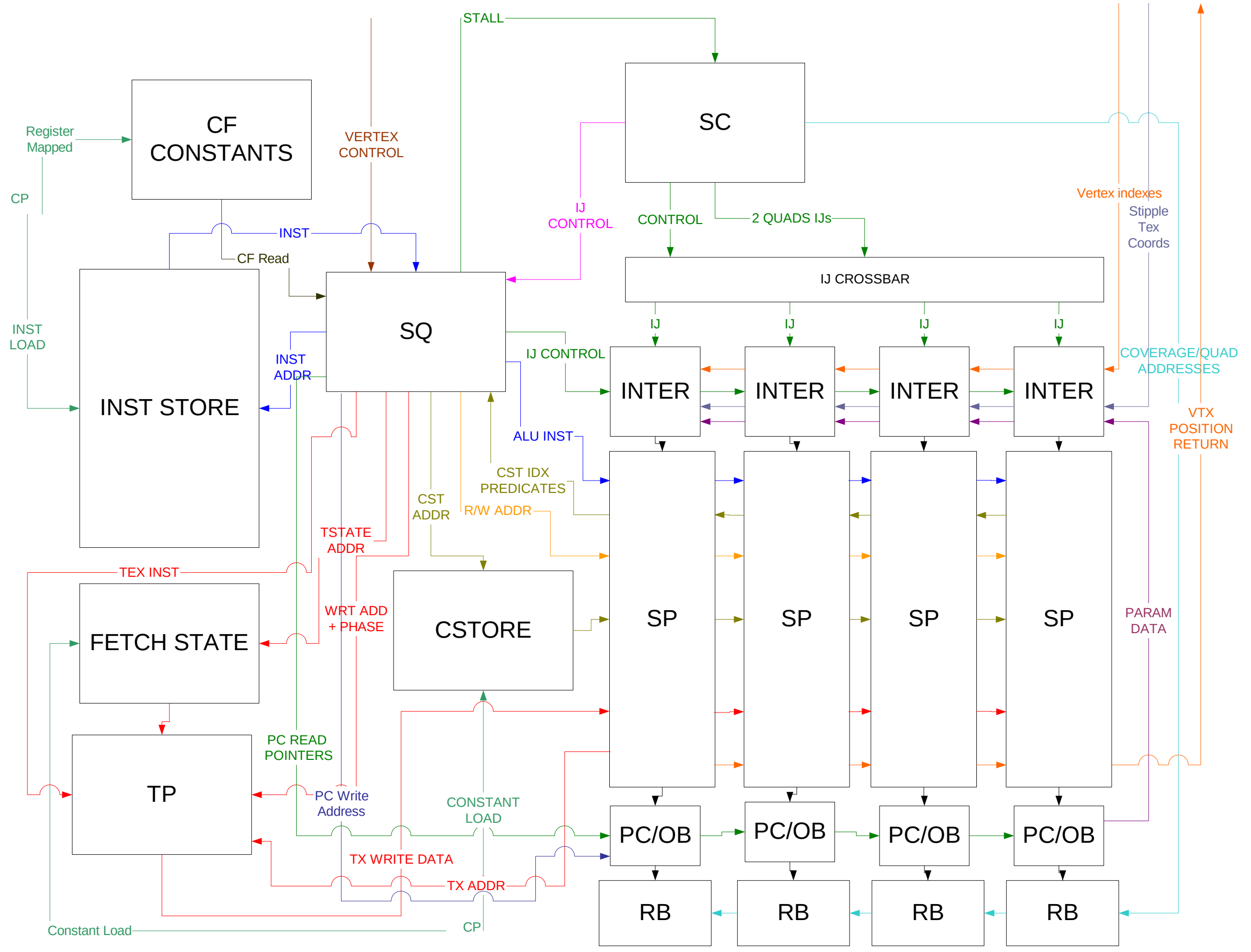


Figure 1: General Sequencer overview

1.1 Top Level Block Diagram

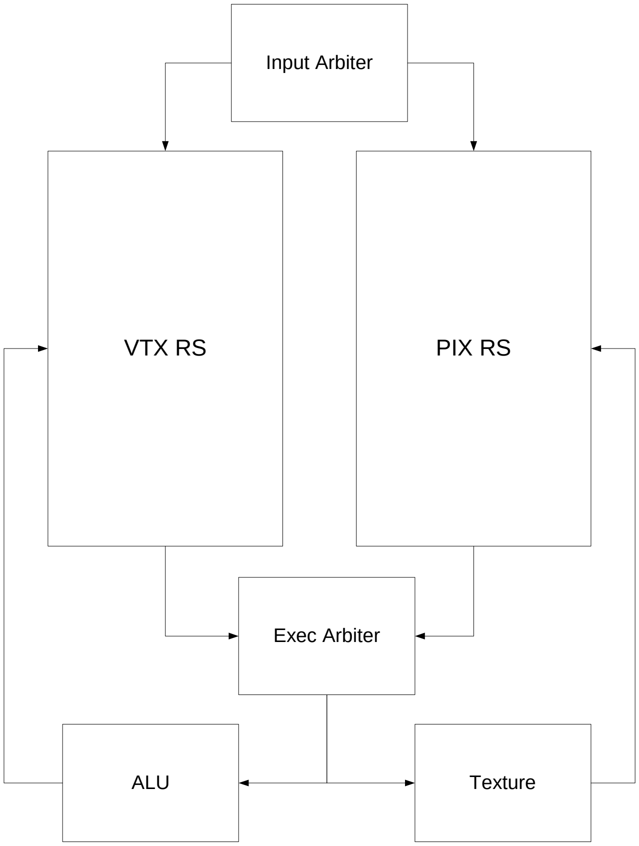


Figure 2: Reservation stations and arbiters

Under this new scheme, the sequencer (SQ) will only use one global state management machine per vector type (pixel, vertex) that we call the reservation station (RS).

1.2 Data Flow graph (SP)

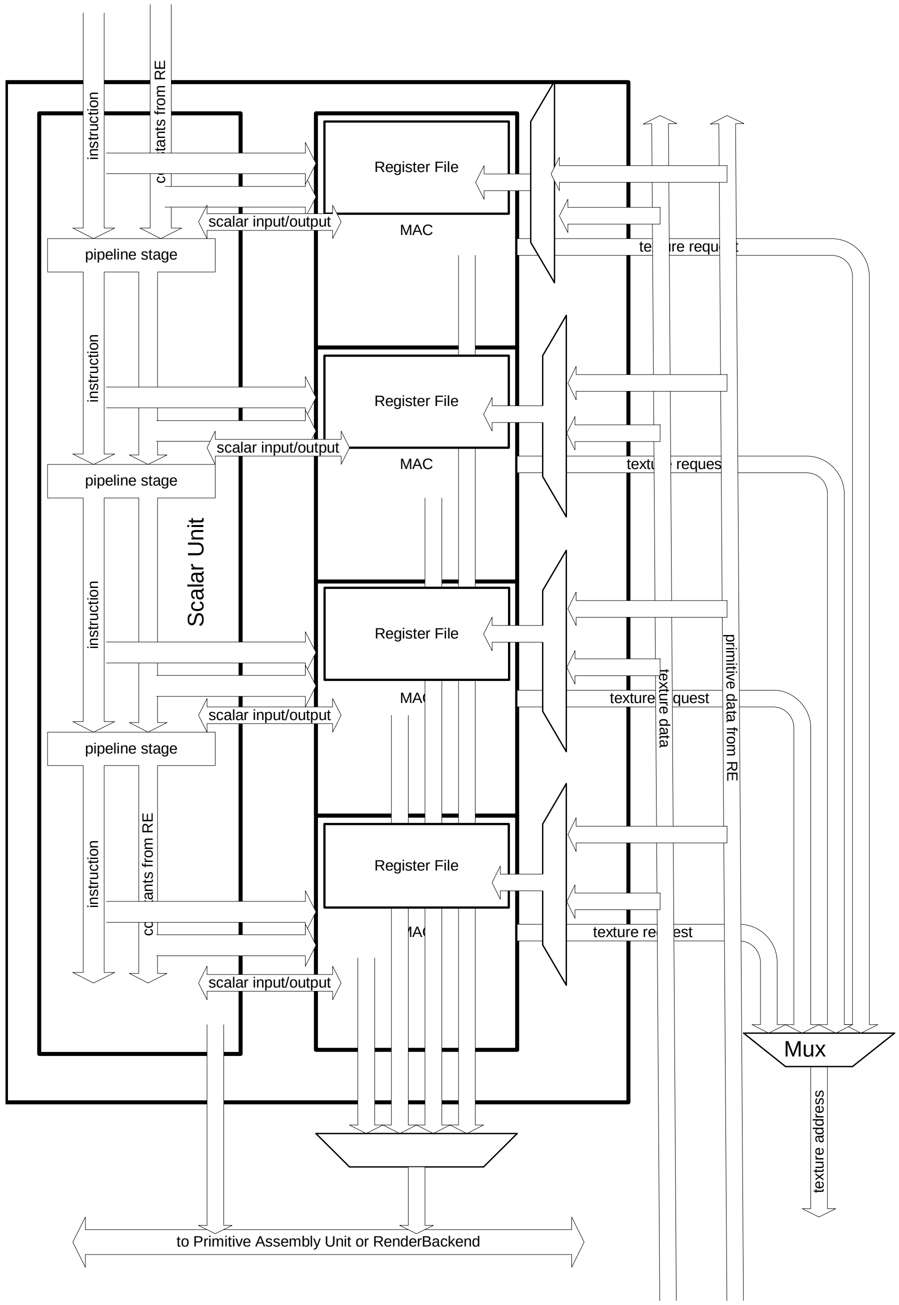


Figure 3: The shader Pipe

The gray area represents blocks that are replicated 4 times per shader pipe (16 times on the overall chip).

1.3 Control Graph

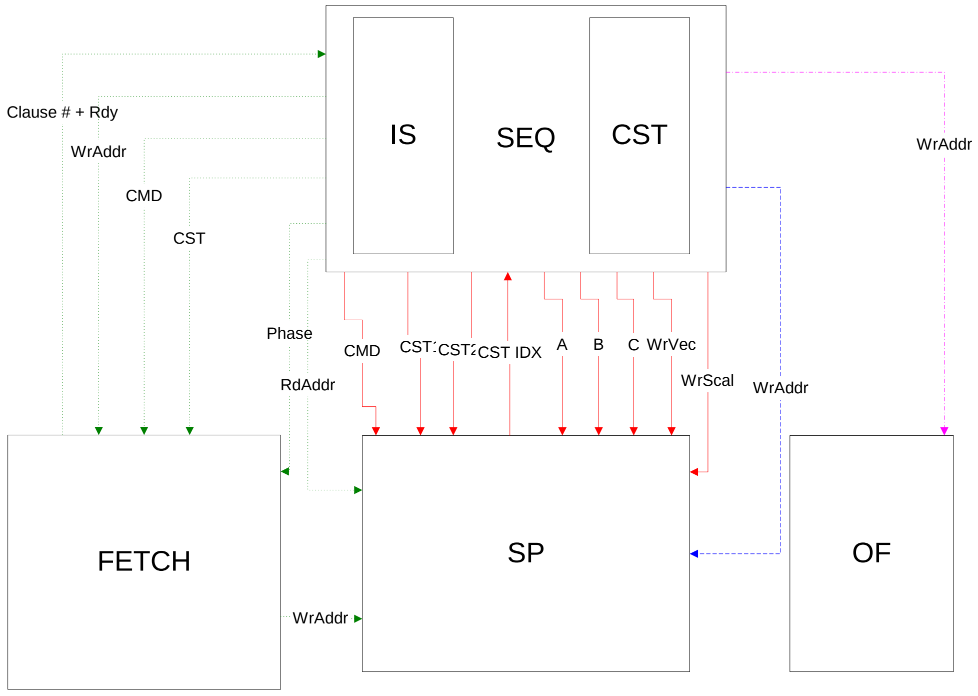


Figure 4: Sequencer Control interfaces

In green is represented the Fetch control interface, in red the ALU control interface, in blue the Interpolated/Vector control interface and in purple is the output file control interface.

2. Interpolated data bus

The interpolators contain an IJ buffer to pack the information as much as possible before writing it to the register file.

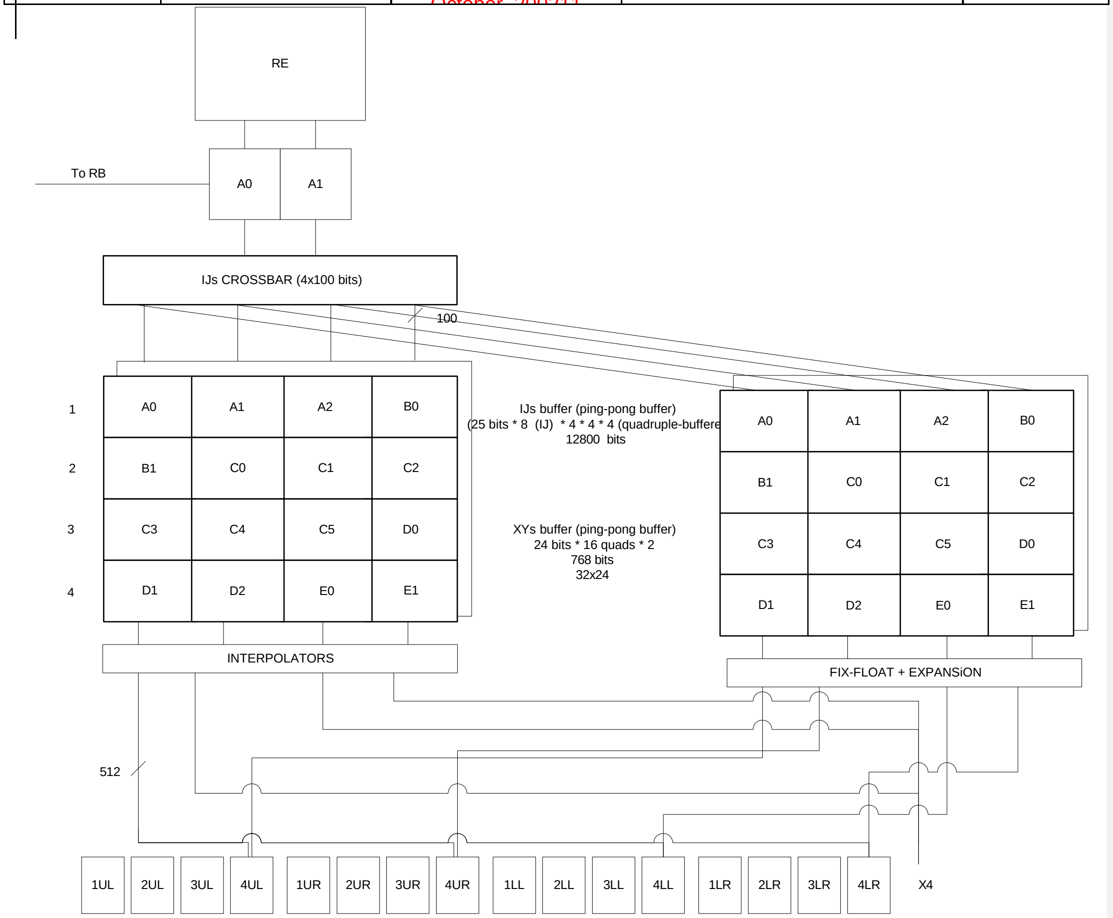
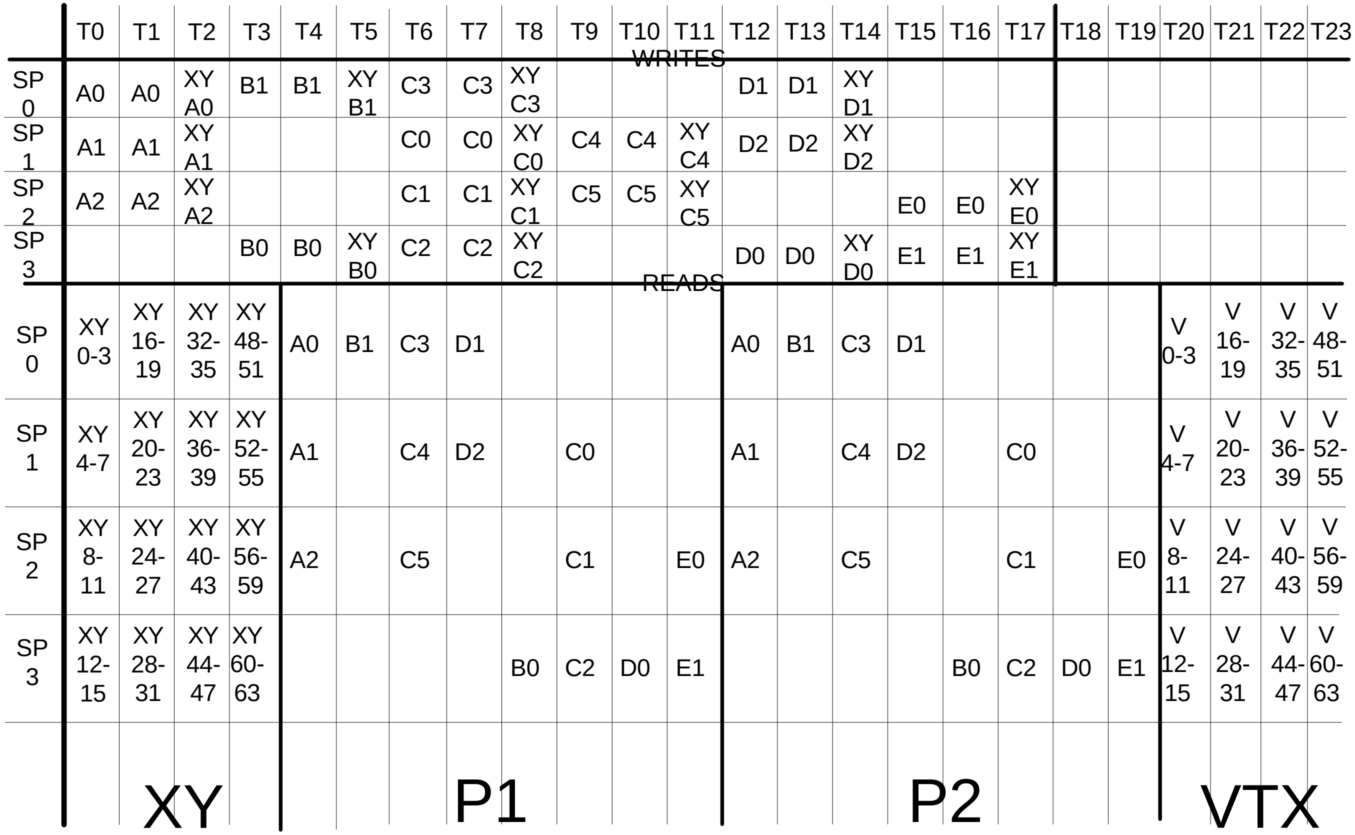


Figure 5: Interpolation buffers



	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21	T22	T23
											WRITES													
SP 0	A0	A0	XY A0	B1	B1	XY B1	C3	C3	XY C3				D1	D1	XY D1									
SP 1	A1	A1	XY A1				C0	C0	XY C0	C4	C4	XY C4	D2	D2	XY D2									
SP 2	A2	A2	XY A2				C1	C1	XY C1	C5	C5	XY C5				E0	E0	XY E0						
SP 3				B0	B0	XY B0	C2	C2	XY C2				D0	D0	XY D0	E1	E1	XY E1						
											READS													
SP 0	XY 0-3	XY 16-19	XY 32-35	XY 48-51	A0	B1	C3	D1					A0	B1	C3	D1					V 0-3	V 16-19	V 32-35	V 48-51
SP 1	XY 4-7	XY 20-23	XY 36-39	XY 52-55	A1		C4	D2		C0			A1		C4	D2		C0			V 4-7	V 20-23	V 36-39	V 52-55
SP 2	XY 8-11	XY 24-27	XY 40-43	XY 56-59	A2		C5			C1		E0	A2		C5			C1		E0	V 8-11	V 24-27	V 40-43	V 56-59
SP 3	XY 12-15	XY 28-31	XY 44-47	XY 60-63					B0	C2	D0	E1					B0	C2	D0	E1	V 12-15	V 28-31	V 44-47	V 60-63
	XY				P1								P2								VTX			

Figure 6: Interpolation timing diagram

Above is an example of a tile the sequencer might receive from the SC. The write side is how the data get stacked into the XY and IJ buffers, the read side is how the data is passed to the GPRs. The IJ information is packed in the IJ buffer 4 quads at a time or two clocks. The sequencer allows at any given time as many as four quads to interpolate a parameter. They all have to come from the same primitive. Then the sequencer controls the write mask to the GPRs to write the valid data in.

3. Instruction Store

There is going to be only one instruction store for the whole chip. It will contain 4096 instructions of 96 bits each.

It is likely to be a 1 port memory; we use 1 clock to load the ALU instruction, 1 clocks to load the Fetch instruction, 1 clock to load 2 control flow instructions and 1 clock to write instructions.

The instruction store is loaded by the CP thru the register mapped registers.

The VS_BASE and PS_BASE context registers are used to specify for each context where its shader is in the instruction memory.

For the Real time commands the story is quite the same but for some small differences. There are no wrap-around points for real time so the driver must be careful not to overwrite regular shader data. The shared code (shared subroutines) uses the same path as real time.

4. Sequencer Instructions

All control flow instructions and move instructions are handled by the sequencer only. The ALUs will perform NOPs during this time (MOV PV,PV, PS,PS) if they have nothing else to do.

5. Constant Stores

5.1 Memory organizations

A likely size for the ALU constant store is 1024x128 bits. The read BW from the ALU constant store is 128 bits/clock and the write bandwidth is 32 bits/clock (directed by the CP bus size not by memory ports).

The maximum logical size of the constant store for a given shader is 256 constants. Or 512 for the pixel/vertex shader pair. The size of the re-mapping table is 128 lines (each line addresses 4 constants). The write granularity is 4 constants or 512 bits. It takes 16 clocks to write the four constants. Real time requires 256 lines in the physical memory (this is physically register mapped).

The texture state is also kept in a similar memory. The size of this memory is 320x96 bits (128 texture states for regular mode, 32 states for RT). The memory thus holds 128 texture states (192 bits per state). The logical size exposes 32 different states total, which are going to be shared between the pixel and the vertex shader. The size of the re-mapping table to for the texture state memory is 32 lines (each line addresses 1 texture state lines in the real memory). The CP write granularity is 1 texture state lines (or 192 bits). The driver sends 512 bits but the CP ignores the top 320 bits. It thus takes 6 clocks to write the texture state. Real time requires 32 lines in the physical memory (this is physically register mapped).

The control flow constant memory doesn't sit behind a renaming table. It is register mapped and thus the driver must reload its content each time there is a change in the control flow constants. Its size is 320*32 because it must hold 8 copies of the 32 dwords of control flow constants and the loop construct constants must be aligned.

The constant re-mapping tables for texture state and ALU constants are logically register mapped for regular mode and physically register mapped for RT operation.

5.2 Management of the Control Flow Constants

The control flow constants are register mapped, thus the CP writes to the according register to set the constant, the SQ decodes the address and writes to the block pointed by its current base pointer (CF_WR_BASE). On the read side, one level of indirection is used. A register (SQ_CONTEXT_MISC.CF_RD_BASE) keeps the current base pointer to the control flow block. This register is copied whenever there is a state change. Should the CP write to CF after the state change, the base register is updated with the (current pointer number +1)% number of states. This way, if the CP doesn't write to CF the state is going to use the previous CF constants.

5.3 Management of the re-mapping tables

5.3.1 *R400 Constant management*

The sequencer is responsible to manage two re-mapping tables (one for the constant store and one for the texture state). On a state change (by the driver), the sequencer will broadside copy the contents of its re-mapping tables to a new one. We have 8 different re-mapping tables we can use concurrently.

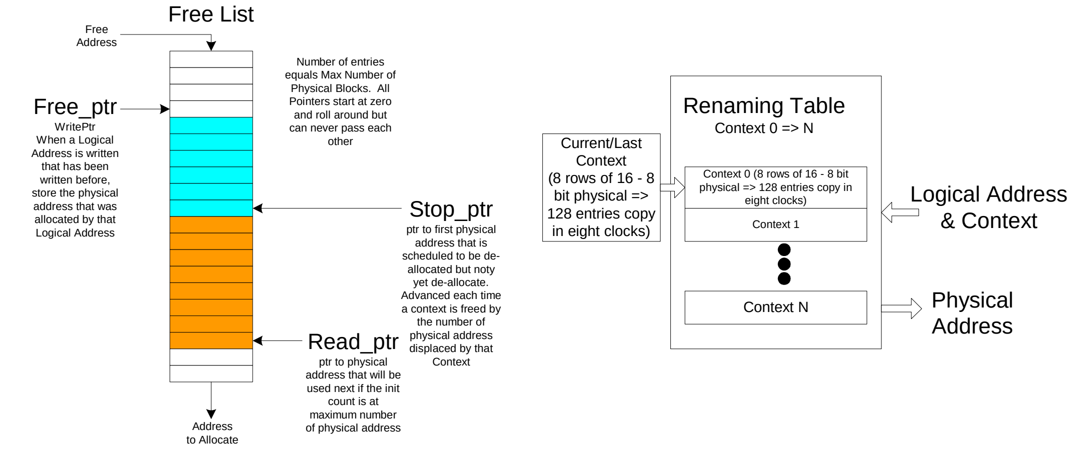
The constant memory update will be incremental, the driver only need to update the constants that actually changed between the two state changes.

For this model to work in its simplest form, the requirement is that the physical memory MUST be at least twice as large as the logical address space + the space allocated for Real Time. In our case, since the logical address space is 512 and the reserved RT space can be up to 256 entries, the memory must be of sizes 1280 and above. Similarly the size of the texture store must be of 32*2+32 = 96 entries and above.

5.3.2 *Proposal for R400LE constant management*

To make this scheme work with only 512+256 = 768 entries, upon reception of a CONTROL packet of state + 1, the sequencer would check for SQ_IDLE and PA_IDLE and if both are idle will erase the content of state to replace it with the new state (this is depicted in Figure 8: De-allocation mechanism~~Figure 8: De-allocation mechanismFigure 8: De-allocation mechanism~~). Note that in the case a state is cleared a value of 0 is written to the corresponding de-allocation counter location so that when the SQ is going to report a state change, nothing will be de-allocated upon the first report.

The second path sets all context dirty bits that were used in the current state to 1 (thus allowing the new state to reuse these physical addresses if needed).


Free List
Free Address
Free_ptr
WritePtr
When a Logical Address is written that has been written before, store the physical address that was allocated by that Logical Address
Number of entries equals Max Number of Physical Blocks. All Pointers start at zero and roll around but can never pass each other
Stop_ptr
ptr to first physical address that is scheduled to be de-allocated but noty yet de-allocate. Advanced each time a context is freed by the number of physical address displaced by that Context
Read_ptr
ptr to physical address that will be used next if the init count is at maximum number of physical address
Address to Allocate
Current/Last Context (8 rows of 16 - 8 bit physical => 128 entries copy in eight clocks)
Renaming Table
Context 0 => N
Context 0 (8 rows of 16 - 8 bit physical => 128 entries copy in eight clocks)
Context 1
Context N
Logical Address & Context
Physical Address

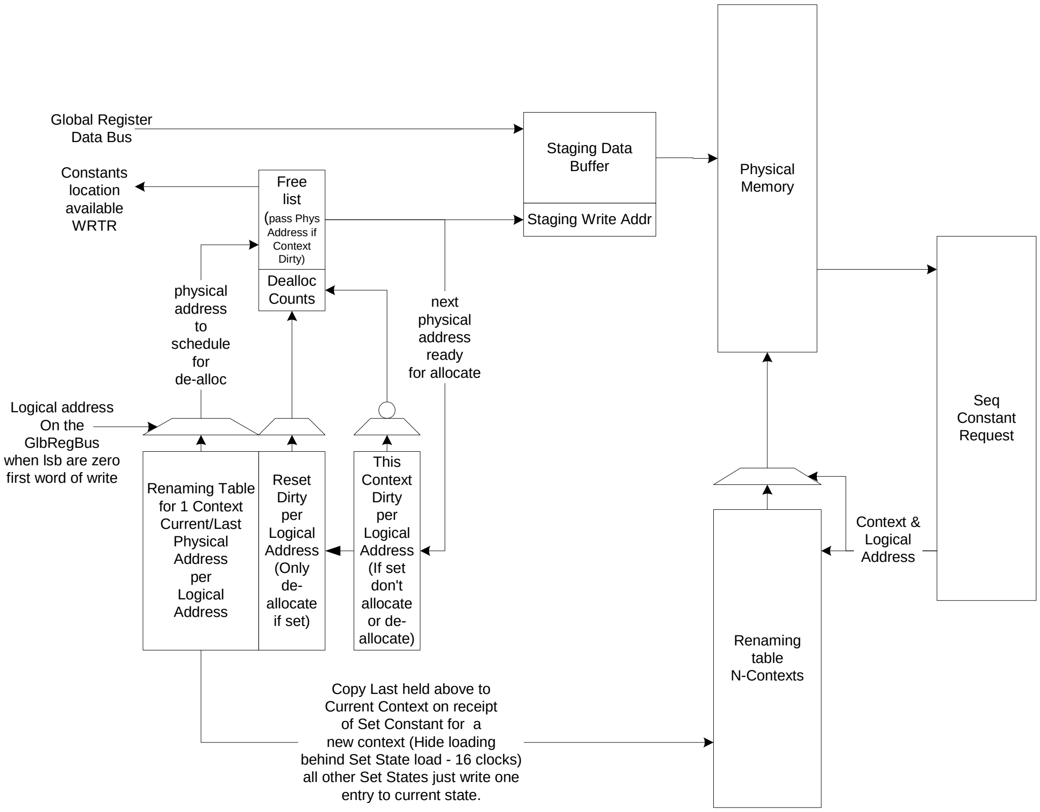

Global Register Data Bus
Staging Data Buffer
Staging Write Addr
Physical Memory
Constants location available WRTR
Free list (pass Phys Address if Context Dirty)
Dealloc Counts
physical address to schedule for de-alloc
next physical address ready for allocate
Logical address On the GlbRegBus when lsb are zero first word of write
Renaming Table for 1 Context Current/Last Physical Address per Logical Address
Reset Dirty per Logical Address (Only de-allocate if set)
This Context Dirty per Logical Address (If set don't allocate or de-allocate)
Seq Constant Request
Context & Logical Address
Renaming table N-Contexts
Copy Last held above to Current Context on receipt of Set Constant for a new context (Hide loading behind Set State load - 16 clocks) all other Set States just write one entry to current state.

Figure 7: Constant management

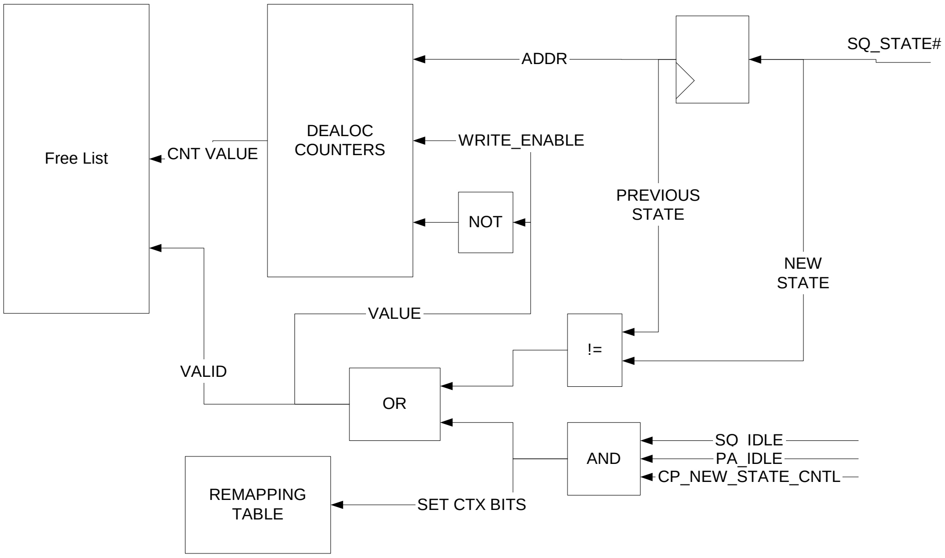


Figure 8: De-allocation mechanism for R400LE

5.3.3 *Dirty bits*

Two sets of dirty bits will be maintained per logical address. The first one will be set to zero on reset and set when the logical address is addressed. The second one will be set to zero whenever a new context is written and set for each address written while in this context. The reset dirty is not set, then writing to that logical address will not require de-allocation of whatever address stored in the renaming table. If it is set and the context dirty is not set, then the physical address store needs to be de-allocated and a new physical address is necessary to store the incoming data. If they are both set, then the data will be written into the physical address held in the renaming for the current logical address. No de-allocation or allocation takes place. This will happen when the driver does a set constant twice to the same logical address between context changes. NOTE: It is important to detect and prevent this, failure to do it will allow multiple writes to allocate all physical memory and thus hang because a context will not fit for rendering to start and thus free up space.

5.3.4 *Free List Block*

A free list block that would consist of a counter (called the IFC or Initial Free Counter) that would reset to zero and incremented every time a chunk of physical memory is used until they have all been used once. This counter would be checked each time a physical block is needed, and if the original ones have not been used up, us a new one, else check the free list for an available physical block address. The count is the physical address for when getting a chunk from the counter.

Storage of a free list big enough to store all physical block addresses.

Maintain three pointers for the free list that are reset to zero. The first one we will call write_ptr. This pointer will identify the next location to write the physical address of a block to be de-allocated. Note: we can never free more physical memory locations than we have. Once recording address the pointer will be incremented to walk the free list like a ring.

The second pointer will be called stop_ptr. The stop_ptr pointer will be advanced by the number of address chunks de-allocates when a context finishes. The address between the stop_ptr and write_ptr cannot be reused because they are still in use. But as soon as the context using then is dismissed the stop_ptr will be advanced.

The third pointer will be called read_ptr. This pointer will point will point to the next address that can be used for allocation as long as the read_ptr does not equal the stop_ptr and the IFC is at its maximum count.

5.3.5 *De-allocate Block*

This block will maintain a free physical address block count for each context. While in current context, a count shall be maintained specifying how many blocks were written into the free list at the write_ptr pointer. This count will be reset upon reset or when this context is active on the back and different than the previous context. It is actually a count of blocks in the previous context that will no longer be used. This count will be used to advance the write_ptr pointer to make available the set of physical blocks freed when the previous context was done. This allows the discard or de-allocation of any number of blocks in one clock.

5.3.6 *Operation of Incremental model*

The basic operation of the model would start with the write_ptr, stop_ptr, read_ptr pointers in the free list set to zero and the free list counter is set to zero. Also all the dirty bits and the previous context will be initialized to zero. When the first set constants happen, the reset dirty bit will not be set, so we will allocate a physical location from the free list counter because its not at the max value. The data will be written into physical address zero. Both the additional copy of the renaming table and the context zeros of the big renaming table will be updated for the logical address that was written by set start with physical address of 0. This process will be repeated for any logical address that are not dirty until the context changes. If a logical address is hit that has its dirty bits set while in the same context, both dirty bits would be set, so the new data will be over-written to the last physical address assigned for this logical address. When the first draw command of the context is detected, the previous context stored in the additional renaming table will be copied to the larger renaming table in the current (new) context location. Then the set constant logical address with be loaded with a new physical address during the copy and if the reset dirty was set, the physical address it replaced in the renaming table would be entered at the write_ptr pointer location on the free list and the write_ptr will be incremented. The de-allocation counter for the previous context (eight) will be incremented. This as set states come in for this context one of the following will happen:

1.) No dirty bits are set for the logical address being updated. A line will be allocated of the free-list counter or the free list at read_ptr pointer if read_ptr != to stop_ptr .
2.) Reset dirty set and Context dirty not set. A new physical address is allocated, the physical address in the renaming table is put on the free list at write_ptr and it is incremented along with the de-allocate counter for the last context.
3.) Context dirty is set then the data will be written into the physical address specified by the logical address.

This process will continue as long as set states arrive. This block will provide backpressure to the CP whenever he has not free list entries available (counter at max and stop_ptr == read_ptr). The command stream will keep a count of contexts of constants in use and prevent more than max constants contexts from being sent.

Whenever a draw packet arrives, the content of the re-mapping table is written to the correct re-mapping table for the context number. Also if the next context uses less constants than the current one all exceeding lines are moved to the free list to be de-allocated later. This happens in parallel with the writing of the re-mapping table to the correct memory.

Now preferable when the constant context leaves the last ALU clause it will be sent to this block and compared with the previous context that left. (Init to zero) If they differ than the older context will no longer be referenced and thus can be de-allocated in the physical memory. This is accomplished by adding the number of blocks freed this context to the stop_ptr pointer. This will make all the physical addresses used by this context available to the read_ptr allocate pointer for future allocation.

This device allows representation of multiple contexts of constants data with N copies of the logical address space. It also allows the second context to be represented as the first set plus some new additional data by just storing the delta's. It allows memory to be efficiently used and when the constants updates are small it can store multiple context. However, if the updates are large, less contexts will be stored and potentially performance will be degraded. Although it will still perform as well as a ring could in this case.

5.4 Constant Store Indexing

In order to do constant store indexing, the sequencer must be loaded first with the indexes (that come from the GPRs). There are 144 wires from the exit of the SP to the sequencer (9 bits pointers x 16 vertexes/clock). Since the data must pass thru the Shader pipe for the float to fixed conversion, there is a latency of 4 clocks (1 instruction)

between the time the sequencer is loaded and the time one can index into the constant store. The assembly will look like this

```
MOVA  R1.X,R2.X        // Loads the sequencer with the content of R2.X, also copies the content of R2.X into R1.X
NOP                    // latency of the float to fixed conversion
ADD   R3,R4,C0[R2.X]// Uses the state from the sequencer to add R4 to C0[R2.X] into R3
```

Note that we don't really care about what is in the brackets because we use the state from the MOVA instruction. R2.X is just written again for the sake of simplicity and coherency.

The storage needed in the sequencer in order to support this feature is 2*64*9 bits = 1152 bits.

5.5 Real Time Commands

The real time commands constants are written by the CP using the register mapped registers allocated for RT. It works is the same way than when dealing with regular constant loads BUT in this case the CP is not sending a logical address but rather a physical address and the reads are not passing thru the re-mapping table but are directly read from the memory. The boundary between the two zones is defined by the CONST_EO_RT control register. Similarly, for the fetch state, the boundary between the two zones is defined by the TSTATE_EO_RT control register.

5.6 Constant Waterfalling

In order to have a reasonable performance in the case of constant store indexing using the address register, we are going to have the possibility of using the physical memory port for read only. This way we can read 1 constant per clock and thus have a worst-case waterfall mode of 1 vertex per clock. There is a small synchronization issue related with this as we need for the SQ to make sure that the constants where actually written to memory (not only sent to the sequencer) before it can allow the first vector of pixels or vertices of the state to go thru the ALUs. To do so, the sequencer keeps 8 bits (one per render state) and sets the bits whenever the last render state is written to memory and clears the bit whenever a state is freed.

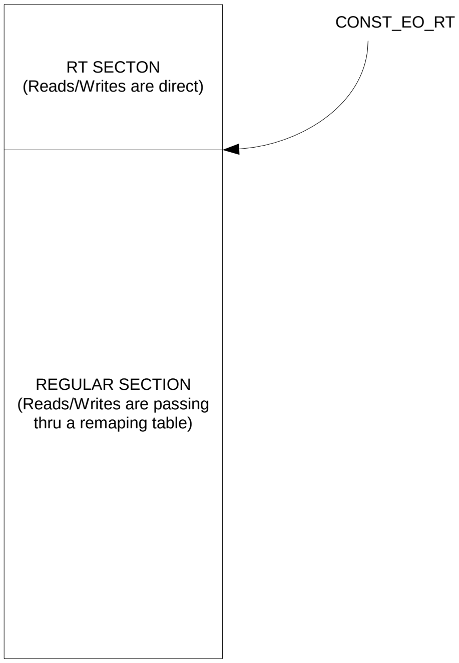


Figure 9: The Constant store

6. Looping and Branches

Loops and branches are planned to be supported and will have to be dealt with at the sequencer level. We plan on supporting constant loops and branches using a control program.

6.1 The controlling state.

The R400 controling state consists of:

Boolean[256:0]
Loop_count[7:0][31:0]
Loop_Start[7:0][31:0]
Loop_Step[7:0][31:0]

That is 256 Booleans and 32 loops.

We have a stack of 4 elements for nested calls of subroutines and 4 loop counters to allow for nested loops.

This state is available on a per shader program basis.

6.2 The Control Flow Program

We'd like to be able to code up a program of the form:

```
1:      Loop
2:      Exec    TexFetch
3:              TexFetch
4:              ALU
5:              ALU
6:              TexFetch
7:      End Loop
8:      ALU Export
```

But realize that 3: may be dependent on 2: and 4: is almost certainly dependent on 2: and 3:. Without clausing, these dependencies need to be expressed in the Control Flow instructions. Additionally, without separate 'texture clauses' and 'ALU clauses' we need to know which instructions to dispatch to the Texture Unit and which to the ALU unit. This information will be encapsulated in the flow control instructions.

Each control flow instruction will contain 2 bits of information for each (non-control flow) instruction:
a) ALU or Texture
b) Serialize Execution

(b) would force the thread to stop execution at this point (before the instruction is executed) and wait until all textures have been fetched. Given the allocation of reserved bits, this would mean that the count of an 'Exec' instruction would be limited to about 8 (non-control-flow) instructions. If more than this were needed, a second Exec (with the same conditions) would be issued.

Another function that relies upon 'clauses' is allocation and order of execution. We need to assure that pixels and vertices are exported in the correct order (even if not all execution is ordered) and that space in the output buffers are allocated in order. Additionally data can't be exported until space is allocated. A new control flow instruction:

Alloc <buffer select -- position,parameter, pixel or vertex memory. And the size required>.

would be created to mark where such allocation needs to be done. To assure allocation is done in order, the actual allocation for a given thread can not be performed unless the equivalent allocation for all previous threads is already completed. The implementation would also assure that execution of instruction(s) following the serialization due to the Alloc will occur in order -- at least until the next serialization or change from ALU to Texture. In most cases this will allow the exports to occur without any further synchronization. Only 'final' allocations or position allocations are

guaranteed to be ordered. Because strict ordering is required for pixels, parameters and positions, this implies only a single alloc for these structures. Vertex exports to memory do not require ordering during allocation and so multiple 'allocs' may be done.

6.2.1 *Control flow instructions table*

Here is the revised control flow instruction set.

Note that whenever a field is marked as RESERVED, it is assumed that all the bits of the field are cleared (0).

NOP		
47 … 44	43	42 … 0
0000	Addressing	RESERVED

This is a regular NOP.

Execute					
47 … 44	43	40 … 34	33 …16	15…12	11 … 0
0001	Addressing	RESERVED	Instructions type + serialize (9 instructions)	Count	Exec Address

Execute_End					
47 … 44	43	40 … 34	33 …16	15…12	11 … 0
0010	Addressing	RESERVED	Instructions type + serialize (9 instructions)	Count	Exec Address

Execute up to 9 instructions at the specified address in the instruction memory. The Instruction type field tells the sequencer the type of the instruction (LSB) (1 = Texture, 0 = ALU and whether to serialize or not the execution (MSB) (1 = Serialize, 0 = Non-Serialized). If Execute_End this is the last execution block of the shader program.

Conditional_Execute						
47 … 44	43	42	41 … 34	33…16	15 …12	11 … 0
0011	Addressing	Condition	Boolean address	Instructions type + serialize (9 instructions)	Count	Exec Address

Conditional_Execute_End						
47 … 44	43	42	41 … 34	33…16	15 …12	11 … 0
0100	Addressing	Condition	Boolean address	Instructions type + serialize (9 instructions)	Count	Exec Address

If the specified Boolean (8 bits can address 256 Booleans) meets the specified condition then execute the specified instructions (up to 9 instructions). If the condition is not met, we go on to the next control flow instruction. If Conditional_Execute_End and the condition is met, this is the last execution block of the shader program.

Conditional_Execute_Predicates							
47 … 44	43	42	41 … 36	35 … 34	33…16	15…12	11 … 0
0101	Addressing	Condition	RESERVED	Predicate vector	Instructions type + serialize (9 instructions)	Count	Exec Address

Conditional_Execute_Predicates_End							
47 … 44	43	42	41 … 36	35 … 34	33…16	15…12	11 … 0
0110	Addressing	Condition	RESERVED	Predicate vector	Instructions type + serialize (9 instructions)	Count	Exec Address

Check the AND/OR of all current predicate bits. If AND/OR matches the condition execute the specified number of instructions. We need to AND/OR this with the kill mask in order not to consider the pixels that aren't valid. If the

condition is not met, we go on to the next control flow instruction. If Conditional_Execute_Predicates_End and the condition is met, this is the last execution block of the shader program.

Conditional_Execute_Predicates_No_Stall							
47 … 44	43	42	41 … 36	35 … 34	33…16	15…12	11 … 0
1101	Addressing	Condition	RESERVED	Predicate vector	Instructions type + serialize (9 instructions)	Count	Exec Address

Conditional_Execute_Predicates_No_Stall_End							
47 … 44	43	42	41 … 36	35 … 34	33…16	15…12	11 … 0
1110	Addressing	Condition	RESERVED	Predicate vector	Instructions type + serialize (9 instructions)	Count	Exec Address

Same as Conditionnal_Execute_Predicates but the SQ is not going to wait for the predicate vector to be updated. You can only set this in the compiler if you know that the predicate set is only a refinement of the current one (like a nested if) because the optimization would still work.

Loop_Start					
47 … 44	43	42 … 21	20 … 16	15…1~~2~~3	12~~1~~ … 0
0111	Addressing	RESERVED	loop ID	RESERVED	Jump address

Loop Start. Compares the loop iterator with the end value. If loop condition not met jump to the address. Forward jump only. Also computes the index value. The loop id must match between the start to end, and also indicates which control flow constants should be used with the loop.

Loop_End						
47 … 44	43	42 … 24	23… 21	20 … 16	15…~~12~~13	~~11~~ 12 … 0
1000	Addressing	RESERVED	Predicate break	loop ID	RESERVED	start address

Loop end. Increments the counter by one, compares the loop count with the end value. If loop condition met, continue, else, jump BACK to the start of the loop. If predicate break != 0, then compares predicate vector n (specified by predicate break number). If all bits cleared then break the loop.

The way this is described does not prevent nested loops, and the inclusion of the loop id make this easy to do.

Conditionnal_Call						
47 … 44	43	42	41 … 34	33 … ~~13~~14	~~12~~13	~~11~~ 12 … 0
1001	Addressing	Condition	Boolean address	RESERVED	Force Call	Jump address

If the condition is met, jumps to the specified address and pushes the control flow program counter on the stack. If force call is set the condition is ignored and the call is made always.

Return		
47 … 44	43	42 … 0
1010	Addressing	RESERVED

Pops the topmost address from the stack and jumps to that address. If nothing is on the stack, the program will just continue to the next instruction.

Conditionnal_Jump							
47 … 44	43	42	41… 34	33	32 … ~~13~~14	~~12~~13	~~11~~ 12 … 0
1011	Addressing	Condition	Boolean address	FW only	RESERVED	Force Jump	Jump address

If force jump is set the condition is ignored and the jump is made always. If FW only is set then only forward jumps are allowed.

Allocate				
47 … 44	43	42…41	40 … 3	2…0
1100	Debug	Buffer Select	RESERVED	Size

Buffer Select takes a value of the following:
01 – position export (ordered export)
10 – parameter cache or pixel export (ordered export)
11 – pass thru (out of order exports).

Size field is only used to reserve space in the export buffer for pass thru exports. Valid values are 1 (1 line) thru 9 (9 lines). It should be determined by the compiler/assembler by taking max index used +1.

If debug is set this is a debug alloc (ignore if debug DB_ON register is set to off).

6.3 Implementation

The envisioned implementation has a buffer that maintains the state of each thread. A thread lives in a given location in the buffer during its entire life, but the buffer has FIFO qualities in that threads leave in the order that they enter. Actually two buffers are maintained -- one for Vertices and one for Pixels. The intended implementation would allow for:

16 entries for vertices
48 entries for pixels.

From each buffer, arbitration logic attempts to select 1 thread for the texture unit and 1 (interleaved) thread for the ALU unit. Once a thread is selected it is read out of the buffer, marked as invalid, and submitted to appropriate execution unit. It is returned to the buffer (at the same place) with its status updated once all possible sequential instructions have been executed. A switch from ALU to TEX or visa-versa or a Serialize_Execution modifier forces the thread to be returned to the buffer.

Each entry in the buffer will be stored across two physical pieces of memory - most bits will be stored in a 1 read port device. Only bits needed for thread arbitration will be stored in a highly multi-ported structure. The bits kept in the 1 read port device will be termed 'state'. The bits kept in the multi-read ported device will be termed 'status'.

'State Bits' needed include:

1. Control Flow Instruction Pointer (13 bits),
2. Execution Count Marker 4 bits),
3. Loop Iterators (4x9 bits),
4. Loop Counters (4x9 bits),

~~4.~~5. Call return pointers (~~4x12~~ 4x13 bits),
~~5.~~6. Predicate Bits (64 bits),
~~6.~~7. Export ID (1 bit),
~~7.~~8. Parameter Cache base Ptr (7 bits),
~~8.~~9. GPR Base Ptr (8 bits),
~~9.~~10. Context Ptr (3 bits).
~~10.~~11. LOD corrections (6x16 bits)
~~11.~~12. Valid bits (64 bits)
~~12.~~13. RT (1 bit) Signifies that this thread is a Real Time thread. This bit must be sent to the Constant store state machine when reading it.

Formatted: Bullets and Numbering

Absent from this list are 'Index' pointers. These are costly enough that I'm presuming that they are instead stored in the GPRs. The first seven fields above (Control Flow Ptr, Execution Count, Loop Counts, call return ptrs, Predicate bits, PC base ptr and export ID) are updated every time the thread is returned to the buffer based on how much progress has been mode on thread execution. GPR Base Ptr, Context Ptr and LOD corrections are unchanged throughout execution of the thread.

'Status Bits' needed include:

- Valid Thread
- Texture/ALU engine needed
- Texture Reads are outstanding
- Waiting on Texture Read to Complete
- Allocation Wait (2 bits)
- 00 – No allocation needed
- 01 – Position export allocation needed (ordered export)
- 10 – Parameter or pixel export needed (ordered export)
- 11 – pass thru (out of order export)
- Allocation Size (4 bits)
- Position Allocated
- Mem/Color Allocated
- First thread of a new context
- Event thread (NULL thread that needs to trickle down the pipe)
- Last (1 bit)
- Pulse SX (1 bit)

All of the above fields from all of the entries go into the arbitration circuitry. The arbitration circuitry will select a winner for both the Texture Engine and for the ALU engine. There are actually two sets of arbitration -- one for pixels and one for vertices. A final selection is then done between the two. But the rest of this implementation summary only considers the 'first' level selection which is similar for both pixels and vertices.

Texture arbitration requires no allocation or ordering so it is purely based on selecting the 'oldest' thread that requires the Texture Engine.

ALU arbitration is a little more complicated. First, only threads where either of Texture_Reads_outstanding or Waiting_on_Texture_Read_to_Complete are '0' are considered. Then if Allocation_Wait is active, these threads are further filtered based on whether space is available. If the allocation is position allocation, then the thread is only considered if all 'older' threads have already done their position allocation (position allocated bits set). If the allocation is parameter or pixel allocation, then the thread is only considered if it is the oldest thread. Also a thread is not considered if it is a parameter or pixel or position allocation, has its First_thread_of_a_new_context bit set and would cause ALU interleaving with another thread performing the same parameter or pixel or position allocation. Finally the 'oldest' of the threads that pass through the above filters is selected. If the thread needed to allocate, then at this time the allocation is done, based on Allocation_Size. If a thread has its “last” bit set, then it is also removed from the buffer, never to return.

If I now redefine 'clauses' to mean 'how many times the thread is removed from the thread buffer for the purpose of exection by either the ALU or Texture engine', then the minimum number of clauses needed is 2 -- one to perform the allocation for exports (execution automatically halts after an 'Alloc' instruction) (but doesn't performs the actual allocation) and one for the actual ALU/export instructions. As the 'Alloc' instruction could be part of a texture clause (presumably the final instruction in such a clause), a thread could still execute in this minimal number of 2 clauses, even if it involved texture fetching.

The Texture_Reads_Outstanding bit must be updated by the sequencer, based on keeping track of how many Texture Clauses have been executed by a given thread that have not yet had there data returned. Any number above 0 results in this bit being set. We could consider forcing synchronization such that two texture clauses for a given thread may not be outstanding at any time (that would be my preference for simplicity reasons and because it would require only very little change in the texture pipe interface). This would allow the sequencer to set the bit on execution of the texture clause, and allow the texture unit to return a pointer to the thread buffer on completion that clears the bit.

6.4 Data dependant predicate instructions

Data dependant conditionals will be supported in the R400. The only way we plan to support those is by supporting three vector/scalar predicate operations of the form:

PRED_SETE_# - similar to SETE except that the result is 'exported' to the sequencer.
PRED_SETNE_# - similar to SETNE except that the result is 'exported' to the sequencer.
PRED_SETGT_# - similar to SETGT except that the result is 'exported' to the sequencer
PRED_SETGTE_# - similar to SETGTE except that the result is 'exported' to the sequencer

For the scalar operations only we will also support the two following instructions:
PRED_SETE0_# – SETE0
PRED_SETE1_# – SETE1

The export is a single bit - 1 or 0 that is sent using the same data path as the MOVA instruction. The sequencer will maintain 4 sets of 64 bit predicate vectors (in fact 8 sets because we interleave two programs but only 4 will be exposed) and use it to control the write masking. This predicate is not maintained across clause boundaries. The # sign is used to specify which predicate set you want to use 0 thru 3.

Then we have two conditional execute bits. The first bit is a conditional execute “on” bit and the second bit tells us if we execute on 1 or 0. For example, the instruction:

P0_ADD_# R0,R1,R2

Is only going to write the result of the ADD into those GPRs whose predicate bit is 0. Alternatively, P1_ADD_# would only write the results to the GPRs whose predicate bit is set. The use of the P0 or P1 without precharging the sequencer with a PRED instruction is undefined.

{Issue: do we have to have a NOP between PRED and the first instruction that uses a predicate?}

6.5 HW Detection of PV,PS

Because of the control program, the compiler cannot detect statically dependant instructions. In the case of non-masked writes and subsequent reads the sequencer will insert uses of PV,PS as needed. This will be done by comparing the read address and the write address of consecutive instructions. For masked writes, the sequencer will insert NOPs wherever there is a dependant read/write.

The sequencer will also have to insert NOPs between PRED_SET and MOVA instructions and their uses.

6.6 Register file indexing

Because we can have loops in fetch clause, we need to be able to index into the register file in order to retrieve the data created in a fetch clause loop and use it into an ALU clause. The instruction will include the base address for register indexing and the instruction will contain these controls:

Bit7	Bit 6	
0	0	'absolute register'
0	1	'relative register'
1	0	'previous vector'
1	1	'previous scalar'

In the case of an absolute register we just take the address as is. In the case of a relative register read we take the base address and we add to it the loop_index and this becomes our new address that we give to the shader pipe.

The sequencer is going to keep a loop index computed as such:

Index = Loop_iterator*Loop_step + Loop_start.

We loop until loop_iterator = loop_count. Loop_step is a signed value [-128…127]. The computed index value is a 10 bit counter that is also signed. Its real range is [-256,256]. The tenth bit is only there so that we can provide an out of range value to the “indexing logic” so that it knows when the provided index is out of range and thus can make the necessary arrangements.

6.7 Debugging the Shaders

In order to be able to debug the pixel/vertex shaders efficiently, we provide 2 methods.

6.7.1 *Method 1: Debugging registers*

Current plans are to expose 2 debugging, or error notification, registers:
1. address register where the first error occurred
2. count of the number of errors

The sequencer will detect the following groups of errors:
- count overflow
- constant indexing overflow
- register indexing overflow

Compiler recognizable errors:
- jump errors
    relative jump address > size of the control flow program
- call stack
    call with stack full
    return with stack empty

A jump error will always cause the program to break. In this case, a break means that a clause will halt execution, but allowing further clauses to be executed.

With all the other errors, program can continue to run, potentially to worst-case limits. The program will only break if the DB_PROB_BREAK register is set.

If indexing outside of the constant or the register range, causing an overflow error, the hardware is specified to return the value with an index of 0. This could be exploited to generate error tokens, by reserving and initializing the 0th register (or constant) for errors.

{ISSUE : Interrupt to the driver or not?}

6.7.2 *Method 2: Exporting the values in the GPRs*

1) The sequencer will have a debug active, count register and an address register for this mode.

Under the normal mode execution follows the normal course.

Under the debug mode it is assumed that the program is always exporting n debug vectors and that all other exports to the SX block (position, color, z, ect) will been turned off (changed into NOPs) by the sequencer (even if they occur before the address stated by the ADDR debug register).

7. Pixel Kill Mask

A vector of 64 bits is kept by the sequencer per group of pixels/vertices. Its purpose is to optimize the texture fetch requests and allow the shader pipe to kill pixels using the following instructions:

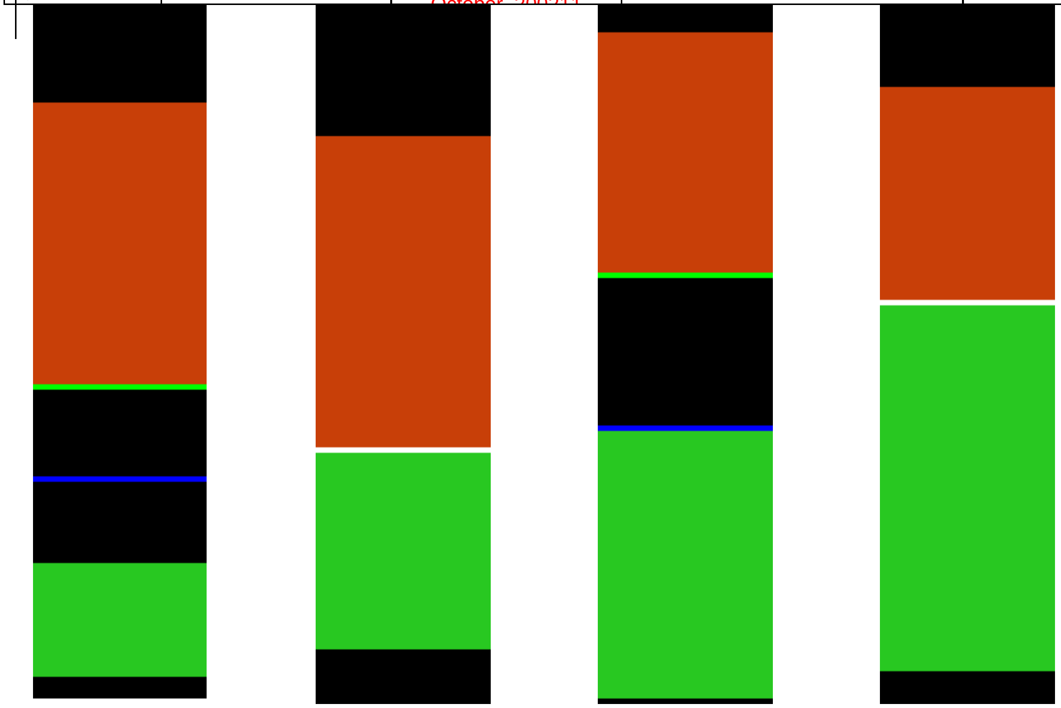
MASK_SETE
MASK_SETNE
MASK_SETGT
MASK_SETGTE

8. Multipass vertex shaders (HOS)

Multipass vertex shaders are able to export from the 6 last clauses but to memory ONLY.

9. Register file allocation

The register file allocation for vertices and pixels can either be static or dynamic. In both cases, the register file in managed using two round robins (one for pixels and one for vertices). In the dynamic case the boundary between pixels and vertices is allowed to move, in the static case it is fixed to 128-VERTEX_REG_SIZE for vertices and PIXEL_REG_SIZE for pixels.



Above is an example of how the algorithm works. Vertices come in from top to bottom; pixels come in from bottom to top. Vertices are in orange and pixels in green. The blue line is the tail of the vertices and the green line is the tail of the pixels. Thus anything between the two lines is shared. When pixels meets vertices the line turns white and the boundary is static until both vertices and pixels share the same “unallocated bubble”. Then the boundary is allowed to move again. The numbering of the GPRs starts from the bottom of the picture at index 0 and goes up to the top at index 127.

10. Fetch Arbitration

The fetch arbitration logic chooses one of the n potentially pending fetch clauses to be executed. The choice is made by looking at the Vs and Ps reservation stations and picking the first one ready to execute. Once chosen, the clause state machine will send one 2x2 fetch per clock (or 4 fetches in one clock every 4 clocks) until all the fetch instructions of the clause are sent. This means that there cannot be any dependencies between two fetches of the same clause.

The arbitrator will not wait for the fetches to return prior to selecting another clause for execution. The fetch pipe will be able to handle up to X(?) in flight fetches and thus there can be a fair number of active clauses waiting for their fetch return data.

11. ALU Arbitration

ALU arbitration proceeds in almost the same way than fetch arbitration. The ALU arbitration logic chooses one of the n potentially pending ALU clauses to be executed. The choice is made by looking at the Vs and Ps reservation stations and picking the first one ready to execute. There are two ALU arbiters, one for the even clocks and one for the odd clocks. For example, here is the sequencing of two interleaved ALU clauses (E and O stands for Even and Odd sets of 4 clocks):

Einst0 Oinst0 Einst1 Oinst1 Einst2 Oinst2 Einst0 Oinst3 Einst1 Oinst4 Einst2 Oinst0…

Proceeding this way hides the latency of 8 clocks of the ALUs. Also note that the interleaving also occurs across clause boundaries.

12. Handling Stalls

When the output file is full, the sequencer prevents the ALU arbitration logic from selecting the last clause (this way nothing can exit the shader pipe until there is place in the output file. If the packet is a vertex packet and the position buffer is full (POS_FULL) then the sequencer also prevents a thread from entering an exporting clause. The sequencer will set the OUT_FILE_FULL signal n clocks before the output file is actually full and thus the ALU arbiter will be able read this signal and act accordingly by not preventing exporting clauses to proceed.

13. Content of the reservation station FIFOs

The reservation FIFOs contain the state of the vector of pixels and vertices. We have two sets of those: one for pixels, and one for vertices. They contain 3 bits of Render State 7 bits for the base address of the GPRs, some bits for LOD correction and coverage mask information in order to fetch fetch for only valid pixels, the quad address.

14. The Output File

The output file is where pixels are put before they go to the RBs. The write BW to this store is 256 bits/clock. Just before this output file are staging registers with write BW 512 bits/clock and read BW 256 bits/clock. The staging registers are 4x128 (and there are 16 of those on the whole chip).

15. IJ Format

The IJ information sent by the PA is of this format on a per quad basis:

We have a vector of IJ's (one IJ per pixel at the centroid of the fragment or at the center of the pixel depending on the mode bit). All pixel's parameters are always interpolated at full 20x24 mantissa precision.

$$P0 = A + I(0)*(B - A) + J(0)*(C - A)$$
$$P1 = A + I(1)*(B - A) + J(1)*(C - A)$$
$$P2 = A + I(2)*(B - A) + J(2)*(C - A)$$
$$P3 = A + I(3)*(B - A) + J(3)*(C - A)$$

P0	P1
P2	P3

Multiplies (Full Precision): 8
Subtracts 19x24 (Parameters): 2
Adds: 8

FORMAT OF P's IJ : Mantissa 20 Exp 4 for I + Sign
Mantissa 20 Exp 4 for J + Sign

Total number of bits : 20*8 + 4*8 + 4*2 = 200.

All numbers are kept using the un-normalized floating point convention: if exponent is different than 0 the number is normalized if not, then the number is un-normalized. The maximum range for the IJs (Full precision) is +/- 1024.

15.1 Interpolation of constant attributes

Because of the floating point imprecision, we need to take special provisions if all the interpolated terms are the same or if two of the terms are the same.

16. Staging Registers

In order for the reuse of the vertices to be 14, the sequencer will have to re-order the data sent IN ORDER by the VGT for it to be aligned with the parameter cache memory arrangement. Given the following group of vertices sent by the VGT:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 || 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 || 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 || 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63

The sequencer will re-arrange them in this fashion:

0 1 2 3 16 17 18 19 32 33 34 35 48 49 50 51 || 4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55 || 8 9 10 11 24 25 26 27 40 41 42 43 56 57 58 59 || 12 13 14 15 28 29 30 31 44 45 46 47 60 61 62 63

The || markers show the SP divisions. In the event a shader pipe is broken, the SQ is responsible to insert padding to account for the missing pipe. For example, if SP1 is broken, vertices 4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55 will not be sent by the VGT to the SQ **AND** the SQ is responsible to “jump” over these vertices in order for no valid vertices to be sent to an invalid SP.

The most straightforward, *non-compressed* interface method would be to convert, in the VGT, the data to 32-bit floating point prior to transmission to the VSISRs. In this scenario, the data would be transmitted to (and stored in) the VSISRs in full 32-bit floating point. This method requires three 24-bit fixed-to-float converters in the VGT. Unfortunately, it also requires and additional 3,072 bits of storage across the VSISRs. This interface is illustrated in Figure 11~~Figure 11Figure 11~~. The area of the fixed-to-float converters and the VSISRs for this method is roughly estimated as 0.759sqmm using the R300 process. The gate count estimate is shown in Figure 10~~Figure 10Figure 10~~.

Basis for 8-deep Latch Memory (from R300)			
8x24-bit	11631 μ^2		60.57813 μ^2 per bit
Area of 96x8-deep Latch Memory	46524 μ^2		
Area of 24-bit Fix-to-float Converter	4712 μ^2 per converter		
Method 1	Block	Quantity	Area
	F2F	3	14136
	8x96 Latch	16	744384
			758520 μ^2

Figure 10:Area Estimate for VGT to Shader Interface

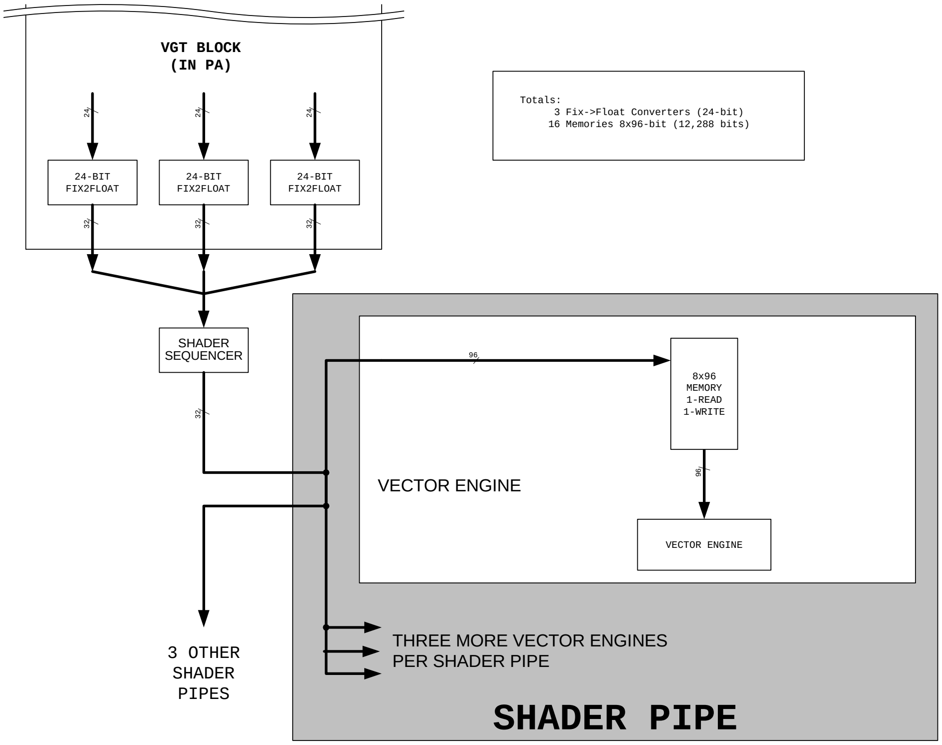


Figure 11:VGT to Shader Interface

17. The parameter cache

The parameter cache is where the vertex shaders export their data. It consists of 16 128x128 memories (1R/1W). The reuse engine will make it so that all vertexes of a given primitive will hit different memories. The allocation method for these memories is a simple round robin. The parameter cache pointers are mapped in the following way: 4MSBs are the memory number and the 7 LSBs are the address within this memory.

MEMORY NUMBER 4 bits	ADDRESS 7 bits

The PA generates the parameter cache addresses as the positions come from the SQ. All it needs to do is keep a Current_Location pointer (7 bits only) and as the positions comes increment the memory number. When the memory number field wraps around, the PA increments the Current_Location by VS_EXPORT_COUNT (a snooped register from the SQ). As an example, say the memories are all empty to begin with and the vertex shader is exporting 8 parameters per vertex (VS_EXPORT_COUNT = 8). The first position received is going to have the PC address 00000000000 the second one 00010000000, third one 00100000000 and so on up to 11110000000. Then the next position received (the 17th) is going to have the address 00000001000, the 18th 00010001000, the 19th 00100001000 and so on. The Current_location is NEVER reset BUT on chip resets. The only thing to be careful about is that if the SX doesn't send you a full group of positions (<64) then you need to fill the address space so that the next group starts correctly aligned (for example if you receive only 33 positions then you need to add 2*VS_EXPORT_COUNT to Current_Location and reset the memory count to 0 before the next vector begins).

17.1 Export restrictions

17.1.1 *Pixel exports:*

Pixels can export 1,2,3 or 4 color buffers to the SX(+z). The exports will be done in order. The PRED_OPTIMIZE function has to be turned of if the exports are done using interleaved predicated instructions. The exports will always be ordered to the SX.

17.1.2 *Vertex exports:*

Position or parameter caches can be exported in any order in the shader program. It is always better to export posistion as soon as possible. Position has to be exported in a single export block (no texture instructions can be placed between the exports). Parameter cache exports can be done in any order with texture instructions interleaved. The PRED_OPTIMIZE function has to be turned of if the exports are done using interleaved predicated instructions to the Parameter cache (see Arbitration restrictions for details). The exports will always be allocated in order to the SX.

17.1.3 *Pass thru exports:*

Pass thru exports have to be done in groups of the form:

```
Alloc 4 (8 or 12)
Execute ALU(ADDR) ALU(DATA) ALU(DATA) ALU(DATA)…
```

They cannot have texture instructions interleaved in the export block. These exports **are not guaranteed to be ordered**.

Also, when doing a pass thru export, Position MUST be exported AFTER all pass thru exports. This position export is used to synchronize the chip when doing a transition from pass thru shader to regular shader and vice versa.

17.2 Arbitration restrictions

Here are the Sequencer arbitration restrictions:

1) Cannot execute a serialized thread if the corresponding texture pending bit is set
2) Cannot allocate position if any older thread has not allocated position
3) If last thread is marked as not valid AND marked as last and we are about to execute the second to oldest thread also marked last then:
    a. Both threads must be from the same context (cannot allow a first thread)
    b. Must turn off the predicate optimization for the second thread
4) Cannot execute a texture clause if texture reads are pending
5) Cannot execute last if texture pending (even if not serial)

18. Export Types

The export type (or the location where the data should be put) is specified using the destination address field in the ALU instruction. Here is a list of all possible export modes:

18.1 Vertex Shading

0:15	- 16 parameter cache
16:31	- Empty (Reserved?)
32	- Export Address
33:37	- 5 vertex exports to the frame buffer and index
38:47	- Empty
48:52	- 5 debug export (interpret as normal memory export)
60	- export addressing mode
61	- Empty
62	- position

63	- sprite size export that goes with position export (X= point size, Y= edge flag is bit 0, Z= VtxKill is bitwise OR of bits 30:0. Any bit other than sign means VtxKill.)

18.2 Pixel Shading

0	- Color for buffer 0 (primary)
1	- Color for buffer 1
2	- Color for buffer 2
3	- Color for buffer 3
4:15	- Empty
16	- Buffer 0 Color/Fog (primary)
17	- Buffer 1 Color/Fog
18	- Buffer 2 Color/Fog
19	- Buffer 3 Color/Fog
20:31	- Empty
32	- Export Address
33:37	- 5 exports for multipass pixel shaders.
38:47	- Empty
48:52	- 5 debug exports (interpret as normal memory export)
60	- export addressing mode
61	- Z for primary buffer (Z exported to 'alpha' component)
62:63	- Empty

19. Special Interpolation modes

19.1 Real time commands

We are unable to use the parameter memory since there is no way for a command stream to write into it. Instead we need to add three 16x128 memories (one for each of three vertices x 16 interpolants). These will be mapped onto the register bus and written by type 0 packets, and output to the the parameter busses (the sequencer and/or PA need to be able to address the realtime parameter memory as well as the regular parameter store. For higher performance we should be able able to view them as two banks of 16 and do double buffering allowing one to be loaded, while the other is rasterized with. Most overlay shaders will need 2 or 4 scalar coordinates, one option might be to restrict the memory to 16x64 or 32x64 allowing only two interpolated scalars per cycle, the only problem I see with this is, if we view support for 16 vector-4 interpolants important (true only if we map Microsoft's high priority stream to the realtime stream), then the PA/sequencer need to support a realtime-specific mode where we need to address 32 vectors of parameters instead of 16. This mode is triggered by the primitive type: REAL TIME. The actual memories are in the in the SX blocks. The parameter data memories are hooked on the RBBM bus and are loaded by the CP using register mapped memory.

19.2 Sprites/ XY screen coordinates/ FB information

XY screen coordinates may be needed in the shader program. This functionality is controlled by the param_gen_I0 register (in SQ) in conjunction with the SND_XY register (in SC) and the param_gen_pos. Also it is possible to send the faceness information (for OGL front/back special operations) to the shader using the same control register. Here is a list of all the modes and how they interact together:

The Data is going to be written in the register specified by the param_gen_pos register.

Param_Gen_I0 disable, snd_xy disable = No modification
Param_Gen_I0 disable, snd_xy enable = No modification
Param_Gen_I0 enable, snd_xy disable = Sign(faceness)garbage,(Sign Point)garbage,Sign(Line)s, t
Param_Gen_I0 enable, snd_xy enable = Sign(faceness)screenX,(Sign Point)screenY,Sign(Line)s, t

In other words,
    The generated vector is (X in RED, Y in GREEN, S in BLUE and T in ALPHA):
    X,Y,S,T

These values are always supposed to be positive and any shader use of them should use the ABS function (as their sign bits will now be used for flags).
SignX = BackFacing
SignY = Point Primitive
SignS = Line Primitive
SignT = currently unused as a flag.

If !Point & !Line, then it is a Poly.

I would assume that one implementation which allows for generic texture lookup (using 3D maps) for poly stipple and AA for the driver would be

```
      if(Y<0) {
            R = 0.0 (Point)
      } else if (S < 0) {
            R = 1.0 (Line)
      } else {
            R = 2.0 (Poly)
}
```

19.3 Auto generated counters

In the cases we are dealing with multipass shaders, the sequencer is going to generate a vector count to be able to both use this count to write the 1st pass data to memory and then use the count to retrieve the data on the 2nd pass. The count is always generated in the same way but it is passed to the shader in a slightly different way depending on the shader type (pixel or vertex). This is toggled on and off using the GEN_INDEX_PIX/VTX register. The sequencer is going to keep two counters, one for pixels and one for vertices. Every time a full vector of vertices or pixels is written to the GPRs the counter is incremented. Every time a RST_PIX_COUNT or RST_VTX_COUNT events are received, the corresponding counter is reset. While there is only one count broadcast to the GPRs, the LSB are hardwired to specific values making the index different for all elements in the vector. Since the count must be different for all pixels/vertices and the 4 LSBs (16 positions) are hardwired to the corresponding shader unit the SQ has two choices:

1) Maintain a 19 bit counter that counts the vectors of 64. In this case the phase must be appended to the count before the count is broadcast to the SPs:

Counter (19 bits)	Phase (2 bits)	Hardwired (4 bits)

2) Maintain a 21 bits counter that counts sub-vectors of 16. In this case only the counter is sent to the Sps:

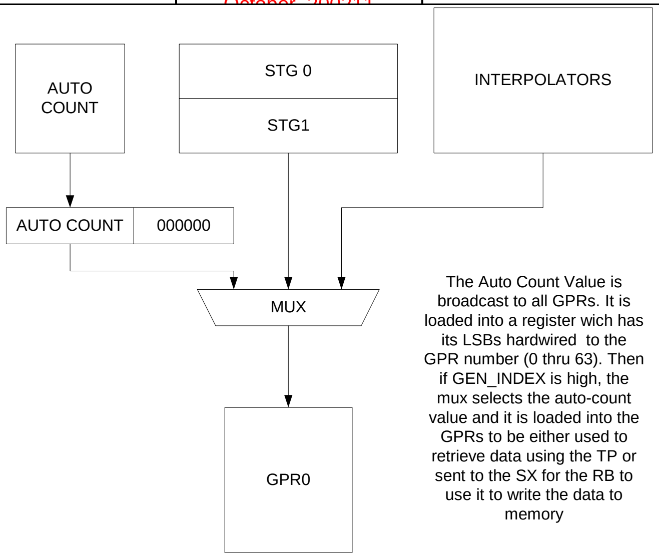
Counter (21 bits)	Hardwired (4 bits)

19.3.1 *Vertex shaders*

In the case of vertex shaders, if GEN_INDEX_VTX is set, the data will be put into the x field of the third register (it means that the compiler must allocate 3 GPRs in all multipass vertex shader modes).

19.3.2 *Pixel shaders*

In the case of pixel shaders, if GEN_INDEX_PIX is set, the data will be put in the x field of the param_gen_pos+1 register.



The Auto Count Value is broadcast to all GPRs. It is loaded into a register wich has its LSBs hardwired to the GPR number (0 thru 63). Then if GEN_INDEX is high, the mux selects the auto-count value and it is loaded into the GPRs to be either used to retrieve data using the TP or sent to the SX for the RB to use it to write the data to memory

Figure 12: GPR input mux Control

20. State management

Every clock, the sequencer will report to the CP the oldest states still in the pipe. These are the states of the programs as they enter the last ALU clause.

20.1 Parameter cache synchronization

In order for the sequencer not to begin a group of pixels before the associated group of vertices has finished, the sequencer will keep a 6 bit count per state (for a total of 8 counters). These counters are initialized to 0 and every time a vertex shader exports its data TO THE PARAMETER CACHE, the corresponding pointer is incremented. When the SC sends a new vector of pixels with the SC_SQ_new_vector bit asserted, the sequencer will first check if the count is greater than 0 before accepting the transmission (it will in fact accept the transmission but then lower its ready to receive). Then the sequencer waits for the count to go to one and decrements it. The sequencer can then issue the group of pixels to the interpolators. Every time the state changes, the new state counter is initialized to 0.

21. XY Address imports

The SC will be able to send the XY addresses to the GPRs. It does so by interleaving the writes of the IJs (to the IJ buffer) with XY writes (to the XY buffer). Then when writing the data to the GPRs, the sequencer is going to interpolate the IJ data or pass the XY data thru a Fix→float converter and expander and write the converted values to the GPRs. The Xys are currently SCREEN SPACE COORDINATES. The values in the XY buffers will wrap. See section 19.2 for details on how to control the interpolation in this mode.

21.1 Vertex indexes imports

In order to import vertex indexes, we have 16 8x96 staging registers. These are loaded one line at a time by the VGT block (96 bits). They are loaded in floating point format and can be transferred in 4 or 8 clocks to the GPRs.

22. Registers

Please see the auto-generated web pages for register definitions.

23. Interfaces

23.1 External Interfaces

Whenever an x is used, it means that the bus is broadcast to all units of the same name. For example, if a bus is named SQ→SPx it means that SQ is going to broadcast the same information to all SP instances.

23.2 SC to SP Interfaces

23.2.1 *SC_SP#*

There is one of these interfaces at front of each of the SP (buffer to stage pixel interpolators). This interface transmits the I,J data for pixel interpolation. For the entire system, two quads per clock are transferred to the 4 SPs, so each of these 4 interfaces transmits one half of a quad per clock. The interface below describes a half of a quad worth of data.
The actual data which is transferred per quad is

Ref Pix I => S4.20 Floating Point I value *4
Ref Pix J => S4.20 Floating Point J value *4

This equates to a total of 200 bits which transferred over 2 clocks
and therefor needs an interface 100 bits wide

Additionally, X,Y data (12-bit unsigned fixed) is conditionally sent across this data bus over the same wires in an additional clock. The X,Y data is sent on the lower 24 bits of the data bus with faceness in the msb.
Transfers across these interfaces are synchronized with the SC_SQ IJ Control Bus transfers.

The data transfer across each of these busses is controlled by a IJ_BUF_INUSE_COUNT in the SC. Each time the SC has sent a pixel vector's worth of data to the SPs, he will increment the IJ_BUF_INUSE_COUNT count. Prior to sending the next pixel vectors data, he will check to make sure the count is less than MAX_BUFER_MINUS_2, if not the SC will stall until the SQ returns a pipelined pulse to decrement the count when he has scheduled a buffer free. Note: We could/may optimize for the case of only sending only IJ to use all the buffers to pre-load more. Currently it is planned for the SP to hold 2 double buffers of I,J data and two buffers of X,Y data, so if either X,Y or Centers and Centroids are on, then the SC can send two Buffers.

In at least the initial version, the SC shall send 16 quads per pixel vector even if the vector is not full. This will increment buffer write address pointers correctly all the time. (We may revisit this for both the SX,SP,SQ and add a EndOfVector signal on all interfaces to quit early. We opted for the simple mode first with a belief that only the end of packet and multiple new vector signals should cause a partial vector and that this would not really be significant performance hit.)

Name	Bits	Description
SC_SP#_data	100	IJ information sent over 2 clocks (or X,Y in 24 LSBs with faceness in upper bit) **Type 0 or 1**, First clock I, second clk J Field ULC URC LLC LRC Bits [63:39] [38:26] [25:13] [12:0] Format SE4M20 SE4M20 SE4M20 SE4M20 **Type 2** Field Face X Y Bits [24] [23:12] [11:0] Format Bit Unsigned Unsigned
SC_SP#_valid	1	Valid
SC_SP#_last_quad_data	1	This bit will be set on the last transfer of data per quad.
SC_SP#_type	2	0 -> Indicates centroids 1 -> Indicates centers 2 -> Indicates X,Y Data and faceness on data bus The SC shall look at state data to determine how many types to send for the

		interpolation process.

The # is included for clarity in the spec and will be replaced with a prefix of u#_ in the verilog module statement for the SC and the SP block will have neither because the instantiation will insert the prefix.

23.2.2 *SC_SQ*

This is the control information sent to the sequencer in order to synchronize and control the interpolation and/or loading data into the GPRs needed to execute a shader program on the sent pixels. This data will be sent over two clocks per transfer with 1 to 16 transfers. Therefore the bus (approx 108 bits) could be folded in half to approx 54 bits.

Name	Bits	Description
SC_SQ_data	46	Control Data sent to the SQ 1 clk transfers Event – valid data consist of event_id and state_id. Instruct SQ to post an event vector to send state id and event_id through request fifo and onto the reservation stations making sure state id and/or event_id gets back to the CP. Events only follow end of packets so no pixel vectors will be in progress. Empty Quad Mask – Transfer Control data consisting of pc_dealloc or new_vector. Receipt of this is to transfer pc_dealloc or new_vector without any valid quad data. New vector will always be posted to request fifo and pc_dealloc will be attached to any pixel vector outstanding or posted in request fifo if no valid quad outstanding. 2 clk transfers Quad Data Valid – Sending quad data with or without new_vector or pc_dealloc. New vector will be posted to request fifo with or without a pixel vector and pc_dealloc will be posted with a pixel vector unless none is in progress. In this case the pc_dealloc will be posted in the request queue. Filler quads will be transferred with The Quad mask set but the pixel corresponding pixel mask set to zero.
SC_SQ_valid	1	SC sending valid data, 2nd clk could be all zeroes

SC_SQ_data – first clock and second clock transfers are shown in the table below.

Name	BitField	Bits	Description
1st Clock Transfer			
SC_SQ_event	0	1	This transfer is a 1 clock event vector Force quad_mask = new_vector=pc_dealloc=0
SC_SQ_event_id	[5:1]	4	This field identifies the event 0 => denotes an End Of State Event 1

			=> TBD
SC_SQ_state_id	[8:6]	3	State/constant pointer (6*3+3)
SC_SQ_pc_dealloc	[11:9]	3	Deallocation token for the Parameter Cache
SC_SQ_new_vector	12	1	The SQ must wait for Vertex shader done count > 0 and after dispatching the Pixel Vector the SQ will decrement the count.
SC_SQ_quad_mask	[16:13]	4	Quad Write mask left to right SP0 => SP3
SC_SQ_end_of_prim	17	1	End Of the primitive
SC_SQ_pix_mask	[33:18]	16	Valid bits for all pixels SP0=>SP3 (UL,UR,LL,LR)
SC_SQ_provok_vtx	[35:34]	2	Provoking vertex for flat shading
SC_SQ_lod_correct_0	[44:36]	9	LOD correction for quad 0 (SP0) (9 bits per quad)
SC_SQ_lod_correct_1	[53:45]	9	LOD correction for quad 1 (SP1) (9 bits per quad)
2nd Clock Transfer			
SC_SQ_lod_correct_2	[8:0]	9	LOD correction for quad 2 (SP2) (9 bits per quad)
SC_SQ_lod_correct_3	[17:9]	9	LOD correction for quad 3 (SP3) (9 bits per quad)
SC_SQ_pc_ptr0	[28:18]	11	Parameter Cache pointer for vertex 0
SC_SQ_pc_ptr1	[39:29]	11	Parameter Cache pointer for vertex 1
SC_SQ_pc_ptr2	[50:40]	11	Parameter Cache pointer for vertex 2
SC_SQ_prim_type	[53:51]	3	Stippled line and Real time command need to load tex cords from alternate buffer 000: Sprite (point) 001: Line 010: Tri_rect 100: Realtime Sprite (point) 101: Realtime Line 110: Realtime Tri_rect

Name	**Bits**	**Description**
SQ_SC_free_buff	1	Pipelined bit that instructs SC to decrement count of buffers in use.
SQ_SC_dec_cntr_cnt	1	Pipelined bit that instructs SC to decrement count of new vector and/or event sent to prevent SC from overflowing SQ interpolator/Reservation request fifo.

The scan converter will submit a partial vector whenever:

1.) He gets a primitive marked with an end of packet signal.
2.) A current pixel vector is being assembled with at least one or more valid quads and the vector has been marked for deallocate when a primitive marked new_vector arrives. The Scan Converter will submit a partial vector (up to 16quads with zero pixel mask to fill out the vector) prior to submitting the new_vector marker\primitive.

(This will prevent a hang which can be demonstrated when all primitives in a packet three vectors are culled except for a one quad primitive that gets marked pc_dealloc (vertices maximum size). In this case two new_vectors are submitted and processed, but then one valid quad with the pc_dealloc creates a vector and then the new would wait for another vertex vector to be processed, but the one being waited for could never export until the pc_dealloc signal made it through and thus the hang.)

23.2.3 SQ to SX(SP): Interpolator bus

Name	Direction	Bits	Description
SQ_SPx_interp_flat_vtx	SQ→SPx	2	Provoking vertex for flat shading
SQ_SPx_interp_flat_gouraud	SQ→SPx	1	Flat or gouraud shading
SQ_SPx_interp_cyl_wrap	SQ→SPx	4	Wich channel needs to be cylindrical wrapped
SQ_SPx_interp_param_gen	SQ→SPx	1	Generate Parameter
SQ_SPx_interp_prim_type	SQ→SPx	2	Bits [1:0] of primitive type sent by SC
SQ_SPx_interp_buff_swap	SQ→SPx	1	Swapp IJ buffers
SQ_SPx_interp_IJ_line	SQ→SPx	2	IJ line number
SQ_SPx_interp_mode	SQ→SPx	1	Center/Centroid sampling
SQ_SXx_pc_ptr0	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_pc_ptr1	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_pc_ptr2	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_rt_sel	SQ→SXx	1	Selects between RT and Normal data (Bit 2 of prim type)
SQ_SX0_pc_wr_en	SQ→SX0	8	Write enable for the PC memories
SQ_SX1_pc_wr_en	SQ→SX1	8	Write enable for the PC memories
SQ_SXx_pc_wr_addr	SQ→SXx	7	Write address for the PCs
SQ_SXx_pc_channel_mask	SQ→SXx	4	Channel mask
SQ_SXx_pc_ptr_valid	SQ→SXx	1	Read pointers are valid.
SQ_SPx_interp_valid	SQ→SPx	1	Interpolation control valid

23.2.4 SQ to SP: Staging Register Data

This is a broadcast bus that sends the VSISR information to the staging registers of the shader pipes.

Name	Direction	Bits	Description
SQ_SPx_vsr_data	SQ→SPx	96	Pointers of indexes or HOS surface information
SQ_SPx_vsr_double	SQ→SPx	1	0: Normal 96 bits per vert 1: double 192 bits per vert
SQ_SP0_ vsr_valid	SQ→SP0	1	Data is valid
SQ_SP1_ vsr_ valid	SQ→SP1	1	Data is valid
SQ_SP2_ vsr_ valid	SQ→SP2	1	Data is valid
SQ_SP3_ vsr_ valid	SQ→SP3	1	Data is valid
SQ_SPx_vsr_read	SQ→SPx	1	Increment the read pointers

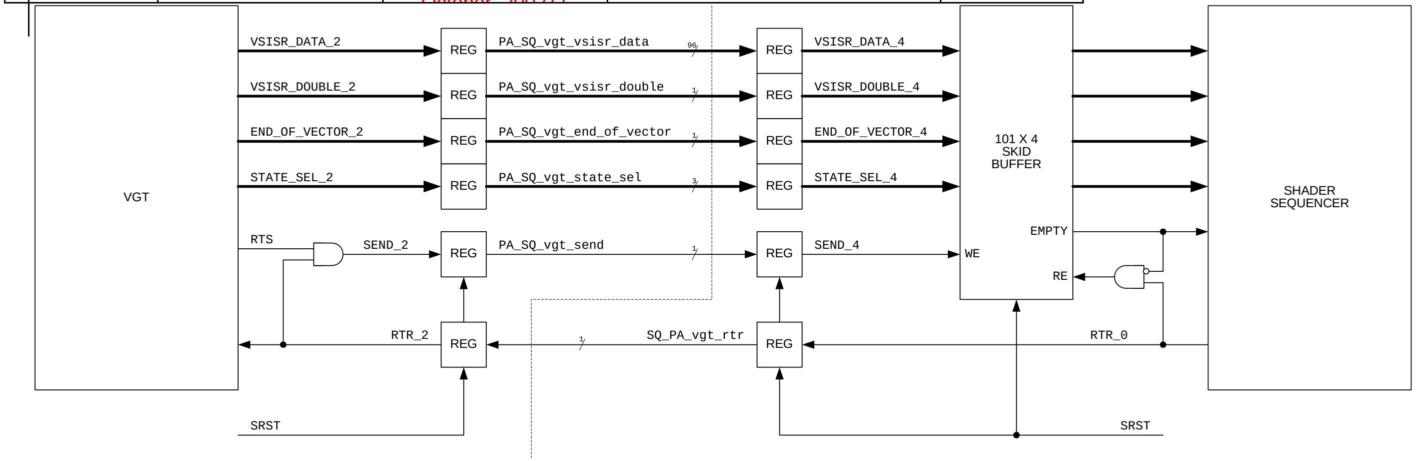
23.2.5 VGT to SQ : Vertex interface

23.2.5.1 Interface Signal Table

The area difference between the two methods is not sufficient to warrant complicating the interface or the state requirements of the VSISRs. **Therefore, the POR for this interface is that the VGT will transmit the data to the VSISRs (via the Shader Sequencer) in full, 32-bit floating-point format.** The VGT can transmit up to six 32-bit floating-point values to each VSISR where four or more values require two transmission clocks. The data bus is 96 bits wide. In the case where an event is sent the 5 LSBs of VGT_SQ_vsisr_data contain the eventID.

Name	Bits	Description
VGT_SQ_vsisr_data	96	Pointers of indexes or HOS surface information
VGT_SQ_event	1	VGT is sending an event
VGT_SQ_vsisr_continued	1	0: Normal 96 bits per vert 1: double 192 bits per vert
VGT_SQ_end_of_vtx_vect	1	Indicates the last VSISR data set for the current process vector (for double vector data, "end_of_vector" is set on the first vector)
VGT_SQ_indx_valid	1	Vsisr data is valid
VGT_SQ_state	3	Render State (6*3+3 for constants). This signal is guaranteed to be correct when “VGT_SQ_vgt_end_of_vector” is high.
VGT_SQ_send	1	Data on the VGT_SQ is valid receive (see write-up for standard R400 SEND/RTR interface handshaking)
SQ_VGT_rtr	1	Ready to receive (see write-up for standard R400 SEND/RTR interface handshaking)

23.2.5.2 Interface Diagrams


VGT
VSISR_DATA_2
VSISR_DOUBLE_2
END_OF_VECTOR_2
STATE_SEL_2
RTS
SEND_2
RTR_2
SRST
REG
PA_SQ_vgt_vsisr_data
PA_SQ_vgt_vsisr_double
PA_SQ_vgt_end_of_vector
PA_SQ_vgt_state_sel
PA_SQ_vgt_send
SQ_PA_vgt_rtr
96
1
1
3
1
1
VSISR_DATA_4
VSISR_DOUBLE_4
END_OF_VECTOR_4
STATE_SEL_4
SEND_4
101 X 4
SKID
BUFFER
EMPTY
WE
RE
RTR_0
SRST
SHADER
SEQUENCER

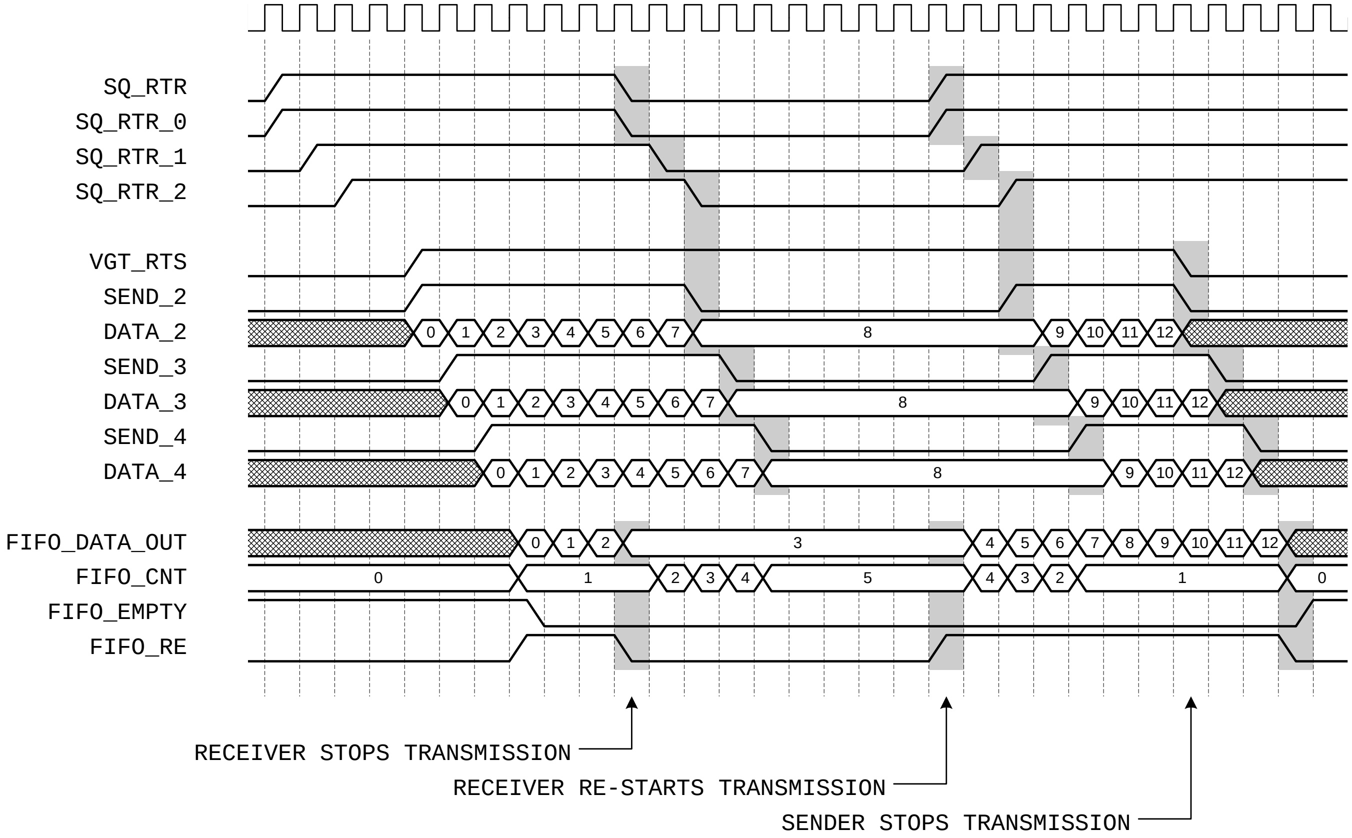


Figure 1. Detailed Logical Diagram for PA_SQ_vgt Interface.

23.2.6 *SQ to SX: Control bus*

Name	**Direction**	**Bits**	**Description**
SQ_SXx_exp_type	SQ→SXx	2	00: Pixel without z (1 to 4 buffers) 01: Pixel with z (1 to 4 buffers) 10: Position (1 or 2 results) 11: Pass thru (4,8 or 12 results aligned)
SQ_SXx_exp_number	SQ→SXx	2	Number of locations needed in the export buffer (encoding depends on the type see bellow).
SQ_SXx_exp_alu_id	SQ→SXx	1	ALU ID
SQ_SXx_exp_valid	SQ→SXx	1	Valid bit
SQ_SXx_exp_state	SQ→SXx	3	State Context
SQ_SXx_free_done	SQ→SXx	1	Pulse that indicates that the previous export is finished **from the point of view of the SP. This does not necessarily mean that the data has been transferred to RB or PA, or that the space in export buffer for that particular vector thread has been freed up.**
SQ_SXx_free_alu_id	SQ→SXx	1	ALU ID

Depending on the type the number of export location changes:

- Type 00 : Pixels without Z
  - 00 = 1 buffer
  - 01 = 2 buffers
  - 10 = 3 buffers
  - 11 = 4 buffer
- Type 01: Pixels with Z
  - 00 = 2 Buffers (color + Z)
  - 01 = 3 buffers (2 color + Z)
  - 10 = 4 buffers (3 color + Z)
  - 11 = 5 buffers (4 color + Z)
- Type 10 : Position export
  - 00 = 1 position
  - 01 = 2 positions
  - 1X = Undefined
- Type 11: Pass Thru
  - 00 = 4 buffers
  - 01 = 8 buffers
  - 10 = 12 buffers
  - 11 = Undefined

Below the thick black line is the end of transfer packet that tells the SX that a given export is finished. The report packet **will always arrive either before or at the same time than the next export to the same ALU id**.

23.2.7 *SX to SQ : Output file control*

Name	**Direction**	**Bits**	**Description**
SXx_SQ_exp_count_rdy	SXx→SQ	1	Raised by SX0 to indicate that the following two fields reflect the result of the most recent export
SXx_SQ_exp_pos_avail	SXx→SQ	2	Specifies whether there is room for another position. 00 : 0 buffers ready 01 : 1 buffer ready 10 : 2 or more buffers ready
SXx_SQ_exp_buf_avail	SXx→SQ	7	Specifies the space available in the output buffers. 0: buffers are full 1: 2K-bits available (32-bits for each of the 64

			pixels in a clause) ... 64: 128K-bits available (16 128-bit entries for each of 64 pixels) 65-127: RESERVED

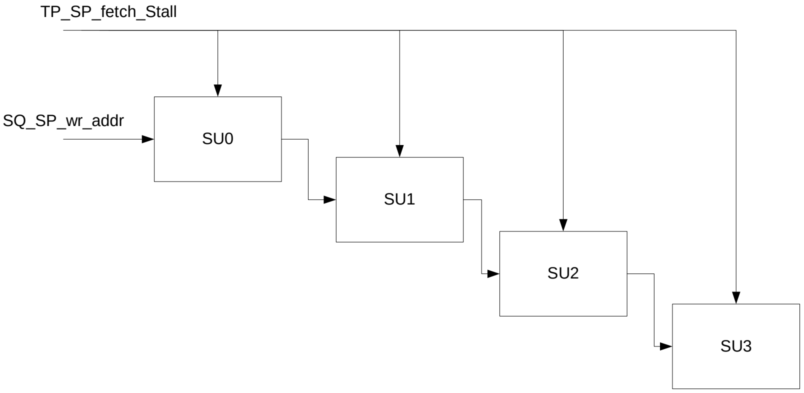
23.2.8 *SQ to TP: Control bus*

Once every clock, the fetch unit sends to the sequencer on which RS line it is now working and if the data in the GPRs is ready or not. This way the sequencer can update the fetch valid bits flags for the reservation station. The sequencer also provides the instruction and constants for the fetch to execute and the address in the register file where to write the fetch return data.

Name	Direction	Bits	Description
TPx_SQ_data_rdy	TPx→ SQ	1	Data ready
TPx_SQ_rs_line_num	TPx→ SQ	6	Line number in the Reservation station
TPx_SQ_type	TPx→ SQ	1	Type of data sent (0:PIXEL, 1:VERTEX)
SQ_TPx_send	SQ→TPx	1	Sending valid data
SQ_TPx_const	SQ→TPx	48	Fetch state sent over 4 clocks (192 bits total)
SQ_TPx_instr	SQ→TPx	24	Fetch instruction sent over 4 clocks
SQ_TPx_end_of_group	SQ→TPx	1	Last instruction of the group
SQ_TPx_Type	SQ→TPx	1	Type of data sent (0:PIXEL, 1:VERTEX)
SQ_TPx_gpr_phase	SQ→TPx	2	Write phase signal
SQ_TP0_lod_correct	SQ→TP0	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP0_pix_mask	SQ→TP0	4	Pixel mask 1 bit per pixel
SQ_TP1_lod_correct	SQ→TP1	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP1_pix_mask	SQ→TP1	4	Pixel mask 1 bit per pixel
SQ_TP2_lod_correct	SQ→TP2	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP2_pix_mask	SQ→TP2	4	Pixel mask 1 bit per pixel
SQ_TP3_lod_correct	SQ→TP3	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP3_pix_mask	SQ→TP3	4	Pixel mask 1 bit per pixel
SQ_TPx_rs_line_num	SQ→TPx	6	Line number in the Reservation station
SQ_TPx_write_gpr_index	SQ->TPx	7	Index into Register file for write of returned Fetch Data
SQ_TPx_ctx_id	SQ→TPx	3	The state context ID (needed for multisample resolves)

23.2.9 *TP to SQ: Texture stall*

The TP sends this signal to the SQ and the SPs when its input buffer is full.



Name	Direction	Bits	Description
TP_SQ_fetch_stall	TP→ SQ	1	Do not send more texture request if asserted

23.2.10 SQ to SP: Texture stall

Name	Direction	Bits	Description
SQ_SPx_fetch_stall	SQ→SPx	1	Do not send more texture request if asserted

23.2.11 SQ to SP: GPR and auto counter

Name	Direction	Bits	Description
SQ_SPx_gpr_wr_addr	SQ→SPx	7	Write address
SQ_SPx_gpr_rd_addr	SQ→SPx	7	Read address
SQ_SPx_gpr_rd_en	SQ→SPx	1	Read Enable
SQ_SP0_gpr_wr_en	SQ→SPx	4	Write Enable for the GPRs of SP0
SQ_SP1_gpr_wr_en	SQ→SPx	4	Write Enable for the GPRs of SP1
SQ_SP2_gpr_wr_en	SQ→SPx	4	Write Enable for the GPRs of SP2
SQ_SP3_gpr_wr_en	SQ→SPx	4	Write Enable for the GPRs of SP3
SQ_SPx_gpr_phase	SQ→SPx	2	The phase mux (arbitrates between inputs, ALU SRC reads and writes)
SQ_SPx_channel_mask	SQ→SPx	4	The channel mask
SQ_SPx_gpr_input_sel	SQ→SPx	2	When the phase mux selects the inputs this tells from which source to read from: Interpolated data, VTX0, VTX1, autogen counter.
SQ_SPx_auto_count	SQ→SPx	21	Auto count generated by the SQ, common for all shader pipes

23.2.12 *SQ to SPx: Instructions*

Name	Direction	Bits	Description
SQ_SPx_instr_start	SQ→SPx	1	Instruction start
SQ_SP_instr	SQ→SPx	24	Transferred over 4 cycles 0: SRC A Negate Argument Modifier 0:0 SRC A Abs Argument Modifier 1:1 SRC A Swizzle 9:2 Vector Dst 15:10 Per channel Select 23:16 00: GPR 01: PV 10: PS 11: Constant (if 11 has to be 11 for all channels) -- 1: SRC B Negate Argument Modifier 0:0 SRC B Abs Argument Modifier 1:1 SRC B Swizzle 9:2 Scalar Dst 15:10 Per channel Select 23:16 00: GPR 01: PV 10: PS 11: Constant (if 11 has to be 11 for all channels) -- 2: SRC C Negate Argument Modifier 0:0 SRC C Abs Argument Modifier 1:1 SRC C Swizzle 9:2 Unused 15:10 Per channel Select 23:16 00: GPR 01: PV 10: PS 11: Constant (if 11 has to be 11 for all channels) -- 3: Vector Opcode 4:0 Scalar Opcode 10:5 Vector Clamp 11:11 Scalar Clamp 12:12 Vector Write Mask 16:13 Scalar Write Mask 20:17 Unused 23:21
SQ_SP0_pred_override	SQ→SP0	4	0: Use per channel RGBA field (enables the per channel logic, if not set only pay attention to the 11 seting). 1: Use GPR
SQ_SP1_pred_override	SQ→SP1	4	0: Use per channel RGBA field (enables the per channel logic, if not set only pay attention to the 11 seting). 1: Use GPR
SQ_SP2_pred_override	SQ→SP2	4	0: Use per channel RGBA field (enables the per channel logic, if not set only pay attention to the 11 seting). 1: Use GPR
SQ_SP3_pred_override	SQ→SP3	4	0: Use per channel RGBA field (enables the per channel logic, if not set only pay attention to the 11 seting). 1: Use GPR
SQ_SPx_exp_id	SQ→SPx	1	GPR ID

SQ_SPx_exporting	SQ→SPx	1	0: Not Exporting 1: Exporting
SQ_SPx_stall	SQ→SPx	1	Stall signal

23.2.13 SQ to SX: write mask interface (must be aligned with the SP data)

Name	Direction	Bits	Description
SQ_SX0_write_mask	SQ→SP0	8	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock. This is for the data coming of SP0 and SP2.
SQ_SX1_ write_mask	SQ→SP1	8	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock. This is for the data coming of SP1 and SP3.

23.2.14 SP to SQ: Constant address load/ Predicate Set/Kill set

Name	Direction	Bits	Description
SP0_SQ_const_addr	SP0→SQ	36	Constant address load / predicate vector load (4 bits only)/ Kill vector load (4 bits only) to the sequencer
SP0_SQ_valid	SP0→SQ	1	Data valid
SP1_SQ_const_addr	SP1→SQ	36	Constant address load / predicate vector load (4 bits only)/ Kill vector load (4 bits only) to the sequencer
SP1_SQ_valid	SP1→SQ	1	Data valid
SP2_SQ_const_addr	SP2→SQ	36	Constant address load / predicate vector load (4 bits only)/ Kill vector load (4 bits only) to the sequencer
SP2_SQ_valid	SP2→SQ	1	Data valid
SP3_SQ_const_addr	SP3→SQ	36	Constant address load / predicate vector load (4 bits only)/ Kill vector load (4 bits only) to the sequencer
SP3_SQ_valid	SP3→SQ	1	Data valid
SP0_SQ_data_type	SP→SQ	2	Data Type 0: Constant Load 1: Predicate Set 2: Kill vector load

Because of the sharing of the bus none of the MOVA, PREDSET or KILL instructions may be coissued.

23.2.15 SQ to SPx: constant broadcast

Name	Direction	Bits	Description
SQ_SPx_const	SQ→SPx	128	Constant broadcast

23.2.16 SQ to CP: RBBM bus

Name	Direction	Bits	Description
SQ_RBB_rs	SQ→CP	1	Read Strobe
SQ_RBB_rd	SQ→CP	32	Read Data
SQ_RBBM_nrtrtr	SQ→CP	1	Optional
SQ_RBBM_rtr	SQ→CP	1	Real-Time (Optional)

23.2.17 CP to SQ: RBBM bus

Name	Direction	Bits	Description
rbbm_we	CP→SQ	1	Write Enable
rbbm_a	CP→SQ	15	Address -- Upper Extent is TBD (16:2)
rbbm_wd	CP→SQ	32	Data
rbbm_be	CP→SQ	4	Byte Enables
rbbm_re	CP→SQ	1	Read Enable
rbb_rs0	CP→SQ	1	Read Return Strobe 0

rbb_rs1	CP→SQ	1	Read Return Strobe 1
rbb_rd0	CP→SQ	32	Read Data 0
rbb_rd1	CP→SQ	32	Read Data 0
RBBM_SQ_soft_reset	CP→SQ	1	Soft Reset

23.2.18 *SQ to CP: State report*

Name	Direction	Bits	Description
SQ_CP_vs_event	SQ→CP	1	Vertex Shader Event
SQ_CP_vs_eventid	SQ→CP	5	Vertex Shader Event ID
SQ_CP_ps_event	SQ→CP	1	Pixel Shader Event
SQ_CP_ps_eventid	SQ→CP	5	Pixel Shader Event ID

23.3 Example of control flow program execution

We now provide some examples of execution to better illustrate the new design.

Given the program:

Alu 0
Alu 1
Tex 0
Tex 1
Alu 3 Serial
Alu 4
Tex 2
Alu 5
Alu 6 Serial
Tex 3
Alu 7
Alloc Position 1 buffer
Alu 8 Export
Tex 4
Alloc Parameter 3 buffers
Alu 9 Export 0
Tex 5
Alu 10 Serial Export 2
Alu 11 Export 1 End

Would be converted into the following CF instructions:

```
Execute 0 Alu 0 Alu 0 Tex 0 Tex 1 Alu 0 Alu 0 Tex 0 Alu 1 Alu 0 Tex
Execute 0 Alu
Alloc Position 1
Execute 0 Alu 0 Tex
Alloc Param 3
Execute_end 0 Alu 0 Tex 1 Alu 0 Alu
```

And the execution of this program would look like this:

Put thread in Vertex RS:

Control Flow Instruction Pointer (12 bits), (CFP)
Execution Count Marker (3 or 4 bits), (ECM)
Loop Iterators (4x9 bits), (LI)
Call return pointers (4x12 bits), (CRP)
Predicate Bits(4x64 bits), (PB)
Export ID (1 bit), (EXID)
GPR Base Ptr (8 bits), (GPR)

Export Base Ptr (7 bits), (EB)
Context Ptr (3 bits).(CPTR)
LOD correction bits (16x6 bits) (LOD)

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	0	0	0	0	0	0	0	0	0

Valid Thread (VALID)
Texture/ALU engine needed (TYPE)
Texture Reads are outstanding (PENDING)
Waiting on Texture Read to Complete (SERIAL)
Allocation Wait (2 bits) (ALLOC)
    00 – No allocation needed
    01 – Position export allocation needed (ordered export)
    10 – Parameter or pixel export needed (ordered export)
    11 – pass thru (out of order export)
Allocation Size (4 bits) (SIZE)
Position Allocated (POS_ALLOC)
First thread of a new context (FIRST)
Last (1 bit), (LAST)

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	0	0	0	0	0	1	0

Then the thread is picked up for the execution of the first control flow instruction:

```
Execute 0 Alu 0 Alu 0 Tex 0 Tex 1 Alu 0 Alu 0 Tex 0 Alu 1 Alu 0 Tex
```

It executes the first two ALU instructions and goes back to the RS for a resource request change. Here is the state returned to the RS:

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	2	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	0	0	0	0	0	1	0

Then when the texture pipe frees up, the arbiter picks up the thread to issue the texture reads. The thread comes back in this state:

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	4	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	1	0	0	0	1	0

Because of the serial bit the arbiter must wait for the texture to return and clear the PENDING bit before it can pick the thread up. Lets say that the texture reads are complete, then the arbiter picks up the thread and returns it in this state:

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	6	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	0	0	0	0	0	1	0

Again the TP frees up, the arbiter picks up the thread and executes. It returns in this state:

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	7	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	0	0	0	1	0

Now, even if the texture has not returned we can still pick up the thread for ALU execution because the serial bit is not set. The thread will however come back to the RS for the second ALU instruction because it has the serial bit set.

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	8	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	1	0	0	0	1	0

As soon as the TP clears the pending bit the thread is picked up and returns:

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	9	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	0	0	0	0	0	1	0

Picked up by the TP and returns:

```
Execute 0 Alu
```

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
1	0	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	0	0	0	1	0

Picked up by the ALU and returns (lets say the TP has not returned yet):

```
Alloc Position 1
```

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
2	0	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	01	1	0	1	0

If the SX has the place for the export, the SQ is going to allocate and pick up the thread for execution. It returns to the RS in this state:

```
Execute 0 Alu 0 Tex
```

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
3	1	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	1	0	0	0	1	1	0

Now, since the TP has not returned yet, we must wait for it to return because we cannot issue multiple texture requests. The TP returns, clears the PENDING bit and we proceed:

```
Alloc Param 3
```

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
4	0	0	0	0	1	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	10	3	1	1	0

Once again the SQ makes sure the SX has enough room in the Parameter cache before it can pick up this thread.

```
Execute_end 0 Alu 0 Tex 1 Alu 0 Alu
```

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
5	1	0	0	0	1	0	100	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	1	0	0	0	1	1	0

This executes on the TP and then returns:

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
5	2	0	0	0	1	0	100	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	1	0	0	1	1	1

Waits for the TP to return because of the textures reads are pending (and SERIAL in this case). Then executes and does not return to the RS because the LAST bit is set. This is the end of this thread and before dropping it on the floor, the SQ notifies the SX of export completion.

24. Open issues

Need to do some testing on the size of the register file as well as on the register file allocation method (dynamic VS static).

Saving power?

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 2015~~16~~ October, 200214	GEN-CXXXXX-REVA	1 of 51

Author: Laurent Lefebvre

Issue To:	**Copy No:**

R400 Sequencer Specification

SQ

Version 2.08~~7~~

Overview: This is an architectural specification for the R400 Sequencer block (SEQ). It provides an overview of the required capabilities and expected uses of the block. It also describes the block interfaces, internal sub-blocks, and provides internal state diagrams.

AUTOMATICALLY UPDATED FIELDS:

Document Location: C:\perforce\r400\doc_lib\design\blocks\sq\R400_Sequencer.doc

Current Intranet Search Title: R400 Sequencer Specification

APPROVALS	
Name/Dept	Signature/Date

Remarks:

THIS DOCUMENT CONTAINS CONFIDENTIAL INFORMATION THAT COULD BE SUBSTANTIALLY DETRIMENTAL TO THE INTEREST OF ATI TECHNOLOGIES INC. THROUGH UNAUTHORIZED USE OR DISCLOSURE.

ATI 2036
LG v. ATI
IPR2015-00326

Table Of Contents

1. OVERVIEW 7
1.1 Top Level Block Diagram 9
1.2 Data Flow graph (SP) 10
1.3 Control Graph 11
2. INTERPOLATED DATA BUS 11
3. INSTRUCTION STORE 14
4. SEQUENCER INSTRUCTIONS 14
5. CONSTANT STORES 14
5.1 Memory organizations 14
5.2 Management of the Control Flow Constants 15
5.3 Management of the re-mapping tables 15
5.3.1 R400 Constant management 15
5.3.2 Proposal for R400LE constant management 15
5.3.3 Dirty bits 17
5.3.4 Free List Block 17
5.3.5 De-allocate Block 18
5.3.6 Operation of Incremental model 18
5.4 Constant Store Indexing 18
5.5 Real Time Commands 19
5.6 Constant Waterfalling 19
6. LOOPING AND BRANCHES 20
6.1 The controlling state. 20
6.2 The Control Flow Program 20
6.2.1 Control flow instructions table 21
6.3 Implementation 23
6.4 Data dependant predicate instructions .... 25~~24~~
6.5 HW Detection of PV,PS .... 26~~25~~
6.6 Register file indexing .... 26~~25~~
6.7 Debugging the Shaders .... 26~~25~~
6.7.1 Method 1: Debugging registers 26
6.7.2 Method 2: Exporting the values in the GPRs .... 27~~26~~
7. PIXEL KILL MASK .... 27~~26~~
8. MULTIPASS VERTEX SHADERS (HOS) .... 27~~26~~
9. REGISTER FILE ALLOCATION .... 27~~26~~
10. FETCH ARBITRATION 28
11. ALU ARBITRATION 28
12. HANDLING STALLS 29
13. CONTENT OF THE RESERVATION STATION FIFOS 29
14. THE OUTPUT FILE 29
15. IJ FORMAT 29
15.1 Interpolation of constant attributes 29
16. STAGING REGISTERS 30

17. THE PARAMETER CACHE 31
17.1 Export restrictions 32
17.1.1 Pixel exports: 32
17.1.2 Vertex exports: 32
17.1.3 Pass thru exports: 32
17.2 Arbitration restrictions 32
18. EXPORT TYPES 32
18.1 Vertex Shading 32
18.2 Pixel Shading 33
19. SPECIAL INTERPOLATION MODES 33
19.1 Real time commands 33
19.2 Sprites/ XY screen coordinates/ FB information 33
19.3 Auto generated counters 34
19.3.1 Vertex shaders 34
19.3.2 Pixel shaders 34
20. STATE MANAGEMENT 35
20.1 Parameter cache synchronization 35
21. XY ADDRESS IMPORTS 35
21.1 Vertex indexes imports 35
22. REGISTERS 35
23. INTERFACES 36
23.1 External Interfaces 36
23.2 SC to SP Interfaces 36
23.2.1 SC_SP# 36
23.2.2 SC_SQ 37
23.2.3 SQ to SX(SP): Interpolator bus 39
23.2.4 SQ to SP: Staging Register Data 39
23.2.5 VGT to SQ : Vertex interface 39
23.2.6 SQ to SX: Control bus 42
23.2.7 SX to SQ : Output file control 42
23.2.8 SQ to TP: Control bus 43
23.2.9 TP to SQ: Texture stall 43
23.2.10 SQ to SP: Texture stall 44
23.2.11 SQ to SP: GPR and auto counter 44
23.2.12 SQ to SPx: Instructions 45
23.2.13 SP to SQ: Constant address load/ Predicate Set/Kill set 46
23.2.14 SQ to SPx: constant broadcast 46
23.2.15 SQ to CP: RBBM bus 46
23.2.16 CP to SQ: RBBM bus 46
23.2.17 SQ to CP: State report 47
23.3 Example of control flow program execution 47

24. OPEN ISSUES 51

Revision Changes:

Rev 0.1 (Laurent Lefebvre) Date: May 7, 2001	First draft.
Rev 0.2 (Laurent Lefebvre) Date : July 9, 2001	Changed the interfaces to reflect the changes in the SP. Added some details in the arbitration section.
Rev 0.3 (Laurent Lefebvre) Date : August 6, 2001	Reviewed the Sequencer spec after the meeting on August 3, 2001.
Rev 0.4 (Laurent Lefebvre) Date : August 24, 2001	Added the dynamic allocation method for register file and an example (written in part by Vic) of the flow of pixels/vertices in the sequencer.
Rev 0.5 (Laurent Lefebvre) Date : September 7, 2001	Added timing diagrams (Vic)
Rev 0.6 (Laurent Lefebvre) Date : September 24, 2001	Changed the spec to reflect the new R400 architecture. Added interfaces.
Rev 0.7 (Laurent Lefebvre) Date : October 5, 2001	Added constant store management, instruction store management, control flow management and data dependant predication.
Rev 0.8 (Laurent Lefebvre) Date : October 8, 2001	Changed the control flow method to be more flexible. Also updated the external interfaces.
Rev 0.9 (Laurent Lefebvre) Date : October 17, 2001	Incorporated changes made in the 10/18/01 control flow meeting. Added a NOP instruction, removed the conditional_execute_or_jump. Added debug registers.
Rev 1.0 (Laurent Lefebvre) Date : October 19, 2001	Refined interfaces to RB. Added state registers.
Rev 1.1 (Laurent Lefebvre) Date : October 26, 2001	Added SEQ→SP0 interfaces. Changed delta precision. Changed VGT→SP0 interface. Debug Methods added.
Rev 1.2 (Laurent Lefebvre) Date : November 16, 2001	Interfaces greatly refined. Cleaned up the spec.
Rev 1.3 (Laurent Lefebvre) Date : November 26, 2001	Added the different interpolation modes.
Rev 1.4 (Laurent Lefebvre) Date : December 6, 2001	Added the auto incrementing counters. Changed the VGT→SQ interface. Added content on constant management. Updated GPRs.
Rev 1.5 (Laurent Lefebvre) Date : December 11, 2001	Removed from the spec all interfaces that weren't directly tied to the SQ. Added explanations on constant management. Added PA→SQ synchronization fields and explanation.
Rev 1.6 (Laurent Lefebvre) Date : January 7, 2002	Added more details on the staging register. Added detail about the parameter caches. Changed the call instruction to a Conditionnal_call instruction. Added details on constant management and updated the diagram.
Rev 1.7 (Laurent Lefebvre) Date : February 4, 2002	Added Real Time parameter control in the SX interface. Updated the control flow section.
Rev 1.8 (Laurent Lefebvre) Date : March 4, 2002	New interfaces to the SX block. Added the end of clause modifier, removed the end of clause instructions.
Rev 1.9 (Laurent Lefebvre) Date : March 18, 2002	Rearangement of the CF instruction bits in order to ensure byte alignement.
Rev 1.10 (Laurent Lefebvre) Date : March 25, 2002	Updated the interfaces and added a section on exporting rules.
Rev 1.11 (Laurent Lefebvre) Date : April 19, 2002	Added CP state report interface. Last version of the spec with the old control flow scheme
Rev 2.0 (Laurent Lefebvre) Date : April 19, 2002	New control flow scheme

Rev 2.01 (Laurent Lefebvre) Date : May 2, 2002	Changed slightly the control flow instructions to allow force jumps and calls.
Rev 2.02 (Laurent Lefebvre) Date : May 13, 2002	Updated the Opcodes. Added type field to the constant/pred interface. Added Last field to the SQ→SP instruction load interface.
Rev 2.03 (Laurent Lefebvre) Date : July 15, 2002	SP interface updated to include predication optimizations. Added the predicate no stall instructions,
Rev 2.04 (Laurent Lefebvre) Date :August 2, 2002	Documented the new parameter generation scheme for XY coordinates points and lines STs.
Rev 2.05 (Laurent Lefebvre) Date : September 10, 2002	Some interface changes and an architectural change to the auto-counter scheme.
Rev 2.06 (Laurent Lefebvre) Date : October 11, 2002	Widened the event interface to 5 bits. Some other little typos corrected.
Rev 2.07 (Laurent Lefebvre) Date : October 14, 2002	Loops, jumps and calls are now using a 13 bit address which allows to jump and call and loop around any control flow addresses (does not requires to be even anymore).
Rev 2.08 (Laurent Lefebvre) Date : October 16, 2002	Clarification updates after discussion with Clay.

1. Overview

The sequencer chooses two ALU threads and a fetch hread to execute, and executes all of the instructions in a block before looking for a new clause of the same type. Two ALU threads are executed interleaved to hide the ALU latency. The arbitrator will give priority to older threads. There are two separate reservation stations, one for pixel vectors and one for vertices vectors. This way a pixel can pass a vertex and a vertex can pass a pixel.

To support the shader pipe the sequencer also contains the shader instruction cache, constant store, control flow constants and texture state. The four shader pipes also execute the same instruction thus there is only one sequencer for the whole chip.

The sequencer first arbitrates between vectors of 64 vertices that arrive directly from primitive assembly and vectors of 16 quads (64 pixels) that are generated in the scan converter.

The vertex or pixel program specifies how many GPRs it needs to execute. The sequencer will not start the next vector until the needed space is available in the GPRs.

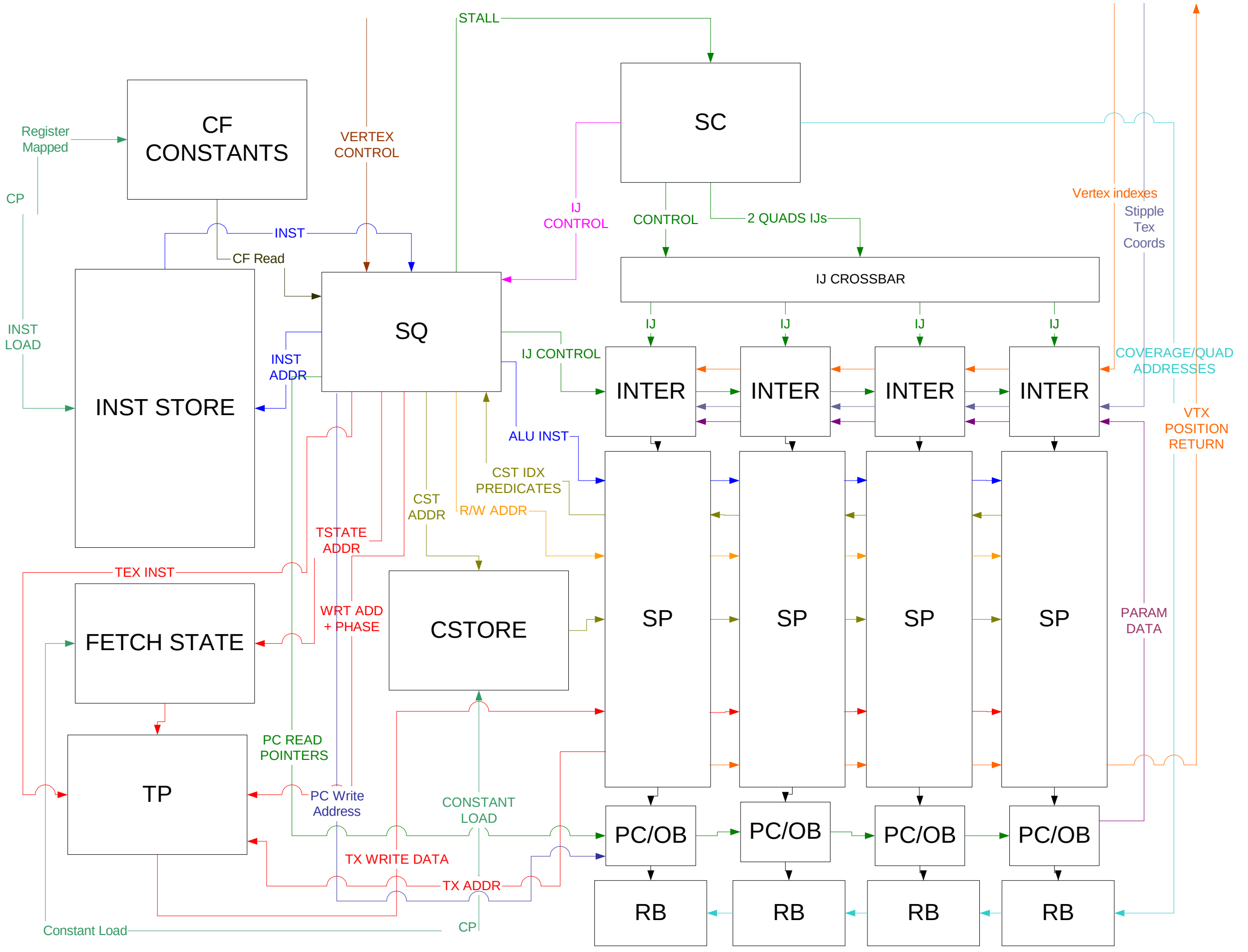


Figure 1: General Sequencer overview

1.1 Top Level Block Diagram

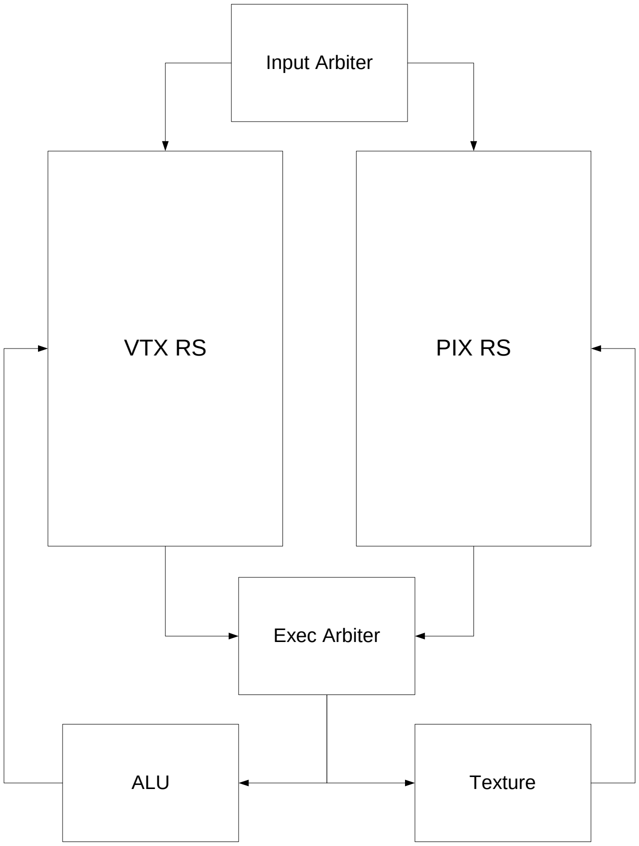


Figure 2: Reservation stations and arbiters

Under this new scheme, the sequencer (SQ) will only use one global state management machine per vector type (pixel, vertex) that we call the reservation station (RS).

1.2 Data Flow graph (SP)

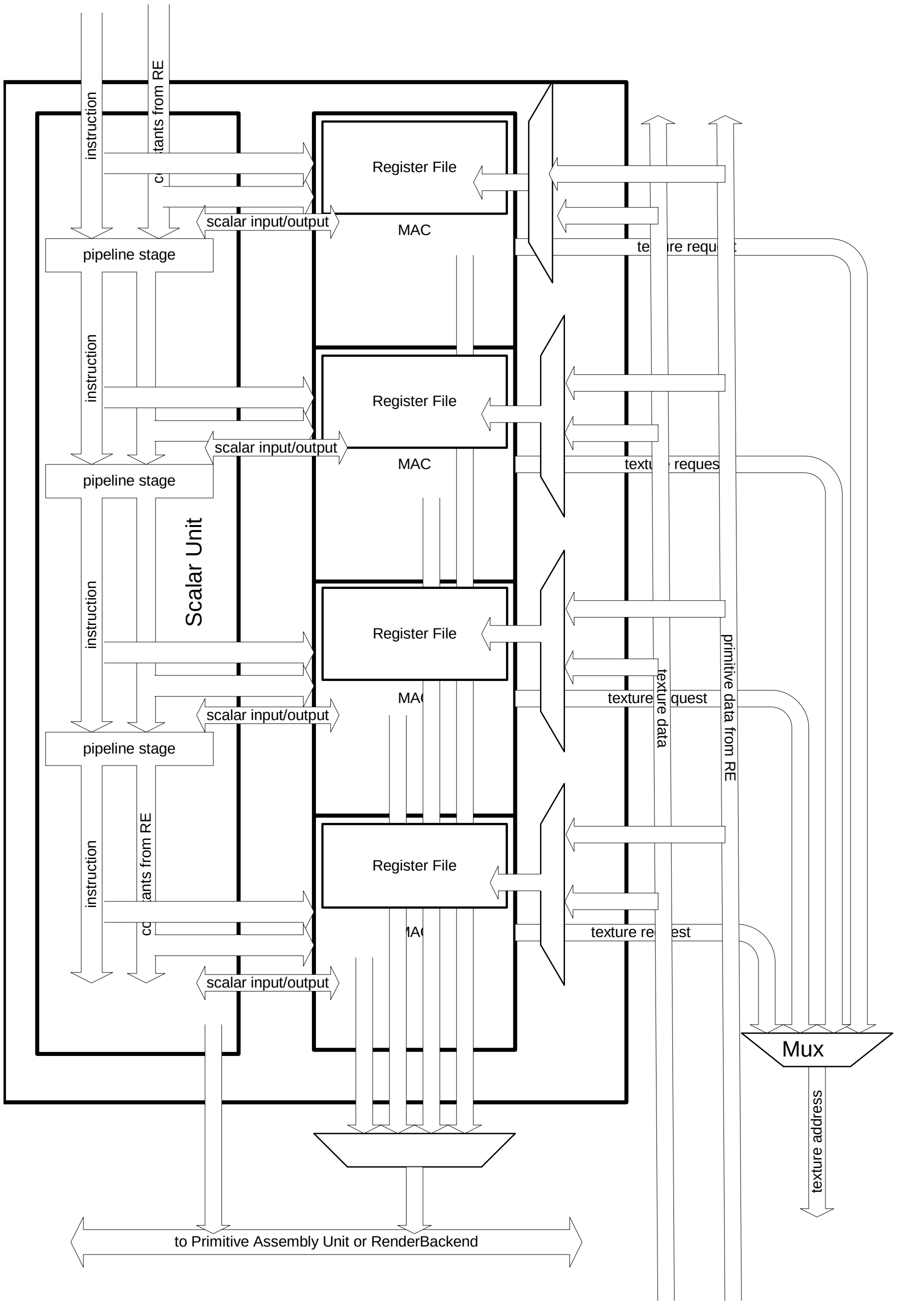


Figure 3: The shader Pipe

The gray area represents blocks that are replicated 4 times per shader pipe (16 times on the overall chip).

1.3 Control Graph

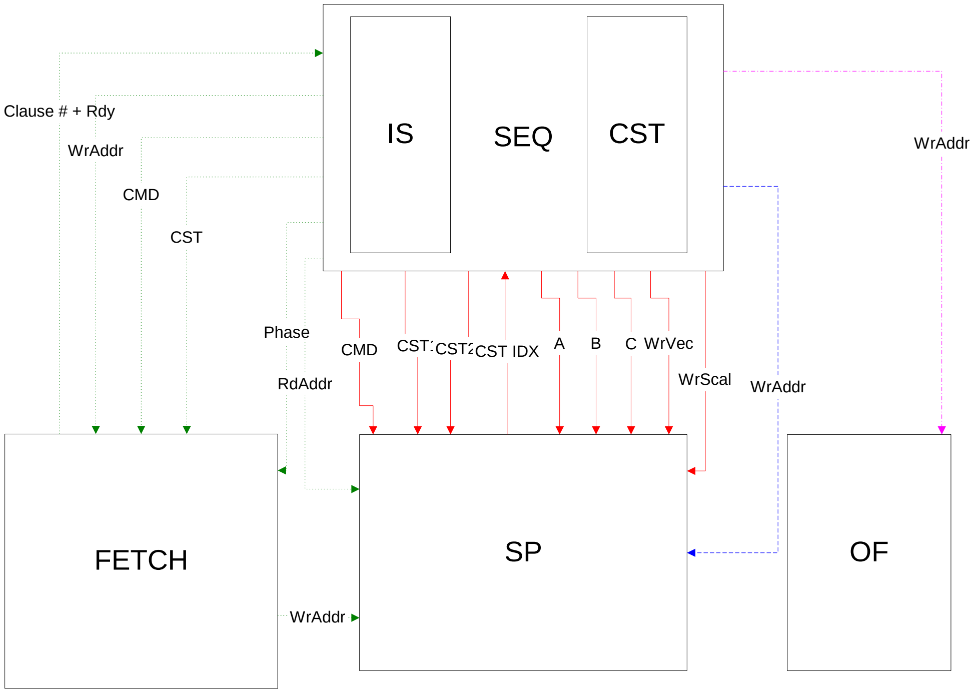


Figure 4: Sequencer Control interfaces

In green is represented the Fetch control interface, in red the ALU control interface, in blue the Interpolated/Vector control interface and in purple is the output file control interface.

2. Interpolated data bus

The interpolators contain an IJ buffer to pack the information as much as possible before writing it to the register file.

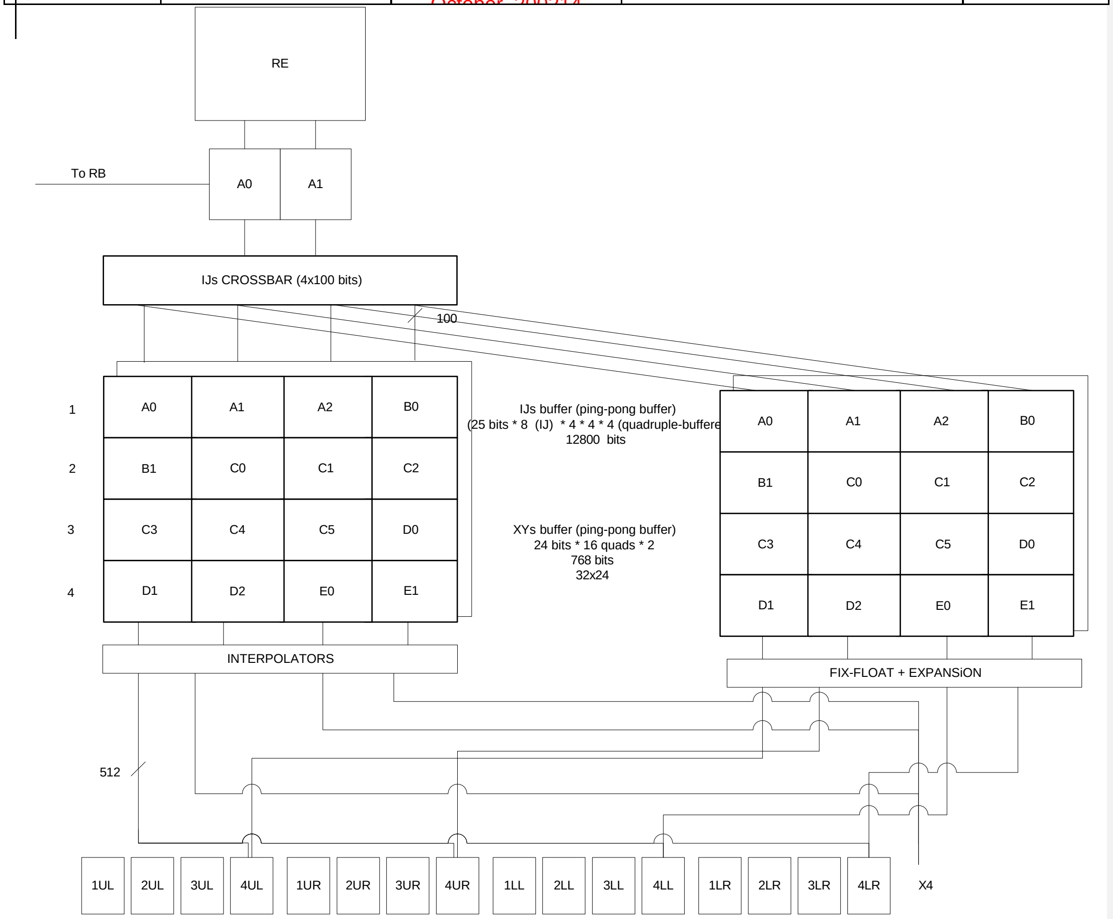


Figure 5: Interpolation buffers

WRITES

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21	T22	T23
SP 0	A0	A0	XY A0	B1	B1	XY B1	C3	C3	XY C3				D1	D1	XY D1									
SP 1	A1	A1	XY A1				C0	C0	XY C0	C4	C4	XY C4	D2	D2	XY D2									
SP 2	A2	A2	XY A2				C1	C1	XY C1	C5	C5	XY C5				E0	E0	XY E0						
SP 3				B0	B0	XY B0	C2	C2	XY C2				D0	D0	XY D0	E1	E1	XY E1						

READS

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21	T22	T23
SP 0	XY 0-3	XY 16-19	XY 32-35	XY 48-51	A0	B1	C3	D1					A0	B1	C3	D1					V 0-3	V 16-19	V 32-35	V 48-51
SP 1	XY 4-7	XY 20-23	XY 36-39	XY 52-55	A1		C4	D2		C0			A1		C4	D2		C0			V 4-7	V 20-23	V 36-39	V 52-55
SP 2	XY 8-11	XY 24-27	XY 40-43	XY 56-59	A2		C5			C1		E0	A2		C5			C1		E0	V 8-11	V 24-27	V 40-43	V 56-59
SP 3	XY 12-15	XY 28-31	XY 44-47	XY 60-63					B0	C2	D0	E1					B0	C2	D0	E1	V 12-15	V 28-31	V 44-47	V 60-63
	XY				P1								P2								VTX			

Figure 6: Interpolation timing diagram

Above is an example of a tile the sequencer might receive from the SC. The write side is how the data get stacked into the XY and IJ buffers, the read side is how the data is passed to the GPRs. The IJ information is packed in the IJ buffer 4 quads at a time or two clocks. The sequencer allows at any given time as many as four quads to interpolate a parameter. They all have to come from the same primitive. Then the sequencer controls the write mask to the GPRs to write the valid data in.

3. Instruction Store

There is going to be only one instruction store for the whole chip. It will contain 4096 instructions of 96 bits each.

It is likely to be a 1 port memory; we use 1 clock to load the ALU instruction, 1 clocks to load the Fetch instruction, 1 clock to load 2 control flow instructions and 1 clock to write instructions.

The instruction store is loaded by the CP thru the register mapped registers.

The VS_BASE and PS_BASE context registers are used to specify for each context where its shader is in the instruction memory.

For the Real time commands the story is quite the same but for some small differences. There are no wrap-around points for real time so the driver must be careful not to overwrite regular shader data. The shared code (shared subroutines) uses the same path as real time.

4. Sequencer Instructions

All control flow instructions ~~and move~~ instructions are handled by the sequencer only. The ALUs will perform NOPs during this time (MOV PV,PV, PS,PS) if they have nothing else to do.

5. Constant Stores

5.1 Memory organizations

A likely size for the ALU constant store is 1024x128 bits. The read BW from the ALU constant store is 128 bits/clock and the write bandwidth is 32 bits/clock (directed by the CP bus size not by memory ports).

The maximum logical size of the constant store for a given shader is 256 constants. Or 512 for the pixel/vertex shader pair. The size of the re-mapping table is 128 lines (each line addresses 4 constants). The write granularity is 4 constants or 512 bits. It takes 16 clocks to write the four constants. Real time requires 256 lines in the physical memory (this is physically register mapped).

The texture state is also kept in a similar memory. The size of this memory is 320x96 bits (128 texture states for regular mode, 32 states for RT). The memory thus holds 128 texture states (192 bits per state). The logical size exposes 32 different states total, which are going to be shared between the pixel and the vertex shader. The size of the re-mapping table to for the texture state memory is 32 lines (each line addresses 1 texture state lines in the real memory). The CP write granularity is 1 texture state lines (or 192 bits). The driver sends 512 bits but the CP ignores the top 320 bits. It thus takes 6 clocks to write the texture state. Real time requires 32 lines in the physical memory (this is physically register mapped).

The control flow constant memory doesn't sit behind a renaming table. It is register mapped and thus the driver must reload its content each time there is a change in the control flow constants. Its size is 320*32 because it must hold 8 copies of the 32 dwords of control flow constants and the loop construct constants must be aligned.

The constant re-mapping tables for texture state and ALU constants are logically register mapped for regular mode and physically register mapped for RT operation.

5.2 Management of the Control Flow Constants

The control flow constants are register mapped, thus the CP writes to the according register to set the constant, the SQ decodes the address and writes to the block pointed by its current base pointer (CF_WR_BASE). On the read side, one level of indirection is used. A register (SQ_CONTEXT_MISC.CF_RD_BASE) keeps the current base pointer to the control flow block. This register is copied whenever there is a state change. Should the CP write to CF after the state change, the base register is updated with the (current pointer number +1)% number of states. This way, if the CP doesn't write to CF the state is going to use the previous CF constants.

5.3 Management of the re-mapping tables

5.3.1 *R400 Constant management*

The sequencer is responsible to manage two re-mapping tables (one for the constant store and one for the texture state). On a state change (by the driver), the sequencer will broadside copy the contents of its re-mapping tables to a new one. We have 8 different re-mapping tables we can use concurrently.

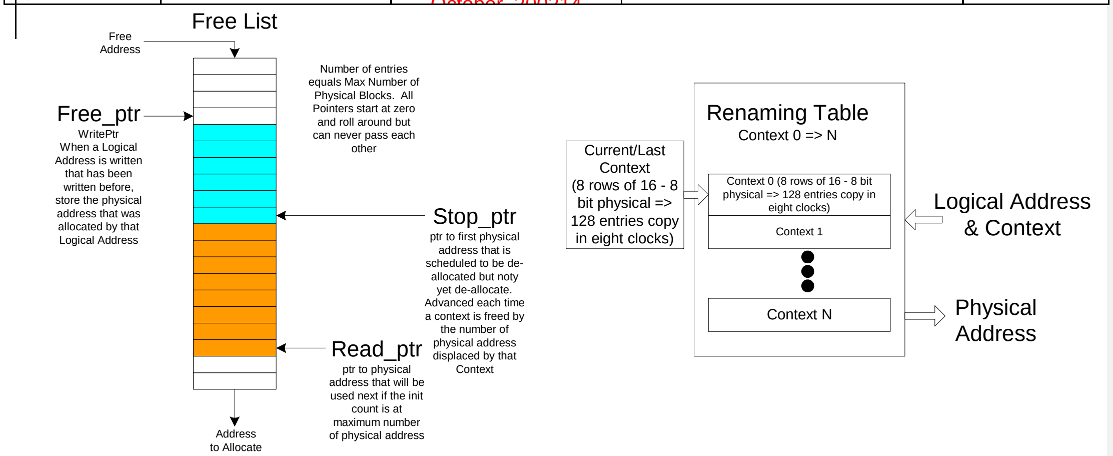
The constant memory update will be incremental, the driver only need to update the constants that actually changed between the two state changes.

For this model to work in its simplest form, the requirement is that the physical memory MUST be at least twice as large as the logical address space + the space allocated for Real Time. In our case, since the logical address space is 512 and the reserved RT space can be up to 256 entries, the memory must be of sizes 1280 and above. Similarly the size of the texture store must be of 32*2+32 = 96 entries and above.

5.3.2 *Proposal for R400LE constant management*

To make this scheme work with only 512+256 = 768 entries, upon reception of a CONTROL packet of state + 1, the sequencer would check for SQ_IDLE and PA_IDLE and if both are idle will erase the content of state to replace it with the new state (this is depicted in Figure 8: De-allocation mechanism~~Figure 8: De-allocation mechanism~~~~Figure 8: De-allocation mechanism~~). Note that in the case a state is cleared a value of 0 is written to the corresponding de-allocation counter location so that when the SQ is going to report a state change, nothing will be de-allocated upon the first report.

The second path sets all context dirty bits that were used in the current state to 1 (thus allowing the new state to reuse these physical addresses if needed).


Free List
Free Address
Free_ptr
WritePtr
When a Logical Address is written that has been written before, store the physical address that was allocated by that Logical Address
Number of entries equals Max Number of Physical Blocks. All Pointers start at zero and roll around but can never pass each other
Stop_ptr
ptr to first physical address that is scheduled to be de-allocated but noty yet de-allocate. Advanced each time a context is freed by the number of physical address displaced by that Context
Read_ptr
ptr to physical address that will be used next if the init count is at maximum number of physical address
Address to Allocate
Current/Last Context (8 rows of 16 - 8 bit physical => 128 entries copy in eight clocks)
Renaming Table
Context 0 => N
Context 0 (8 rows of 16 - 8 bit physical => 128 entries copy in eight clocks)
Context 1
Context N
Logical Address & Context
Physical Address

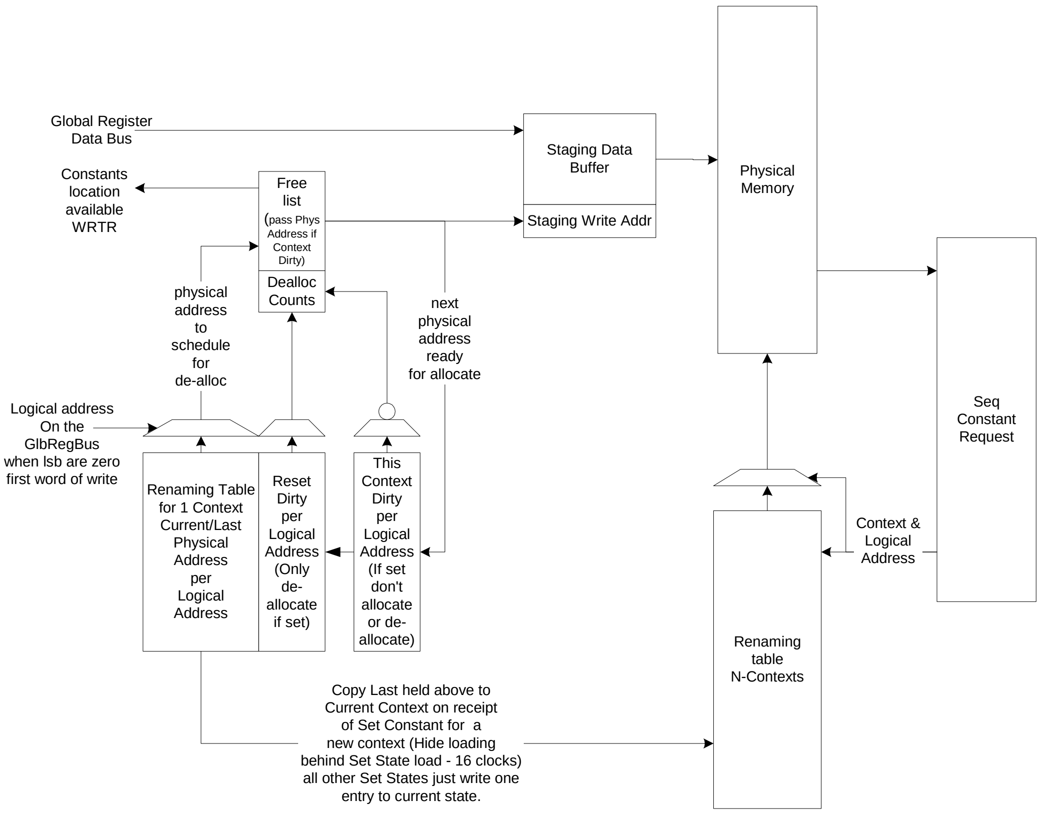

Global Register Data Bus
Staging Data Buffer
Staging Write Addr
Physical Memory
Constants location available WRTR
Free list (pass Phys Address if Context Dirty)
Dealloc Counts
physical address to schedule for de-alloc
next physical address ready for allocate
Seq Constant Request
Logical address On the GlbRegBus when lsb are zero first word of write
Renaming Table for 1 Context Current/Last Physical Address per Logical Address
Reset Dirty per Logical Address (Only de-allocate if set)
This Context Dirty per Logical Address (If set don't allocate or de-allocate)
Context & Logical Address
Renaming table N-Contexts
Copy Last held above to Current Context on receipt of Set Constant for a new context (Hide loading behind Set State load - 16 clocks) all other Set States just write one entry to current state.

Figure 7: Constant management

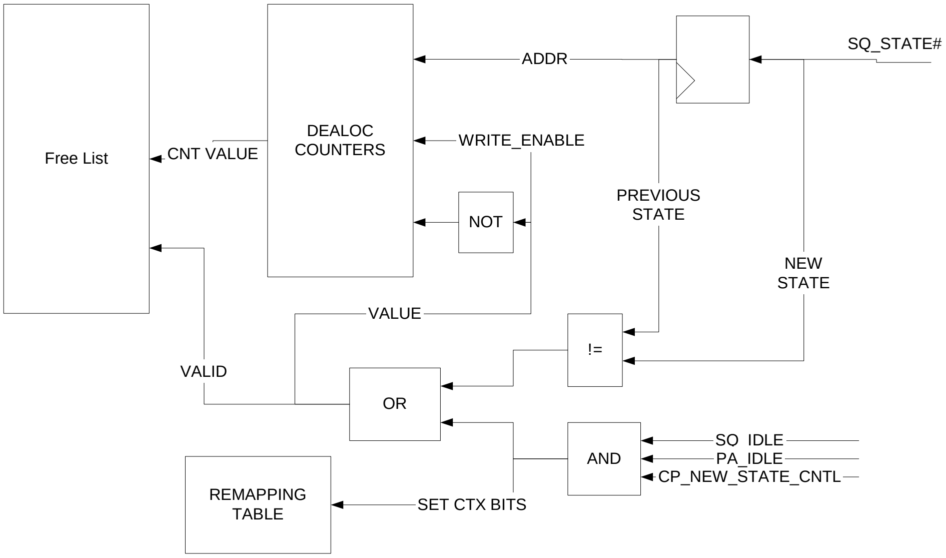


Figure 8: De-allocation mechanism for R400LE

5.3.3 *Dirty bits*

Two sets of dirty bits will be maintained per logical address. The first one will be set to zero on reset and set when the logical address is addressed. The second one will be set to zero whenever a new context is written and set for each address written while in this context. The reset dirty is not set, then writing to that logical address will not require de-allocation of whatever address stored in the renaming table. If it is set and the context dirty is not set, then the physical address store needs to be de-allocated and a new physical address is necessary to store the incoming data. If they are both set, then the data will be written into the physical address held in the renaming for the current logical address. No de-allocation or allocation takes place. This will happen when the driver does a set constant twice to the same logical address between context changes. NOTE: It is important to detect and prevent this, failure to do it will allow multiple writes to allocate all physical memory and thus hang because a context will not fit for rendering to start and thus free up space.

5.3.4 *Free List Block*

A free list block that would consist of a counter (called the IFC or Initial Free Counter) that would reset to zero and incremented every time a chunk of physical memory is used until they have all been used once. This counter would be checked each time a physical block is needed, and if the original ones have not been used up, us a new one, else check the free list for an available physical block address. The count is the physical address for when getting a chunk from the counter.

Storage of a free list big enough to store all physical block addresses.

Maintain three pointers for the free list that are reset to zero. The first one we will call write_ptr. This pointer will identify the next location to write the physical address of a block to be de-allocated. Note: we can never free more physical memory locations than we have. Once recording address the pointer will be incremented to walk the free list like a ring.

The second pointer will be called stop_ptr. The stop_ptr pointer will be advanced by the number of address chunks de-allocates when a context finishes. The address between the stop_ptr and write_ptr cannot be reused because they are still in use. But as soon as the context using then is dismissed the stop_ptr will be advanced.

The third pointer will be called read_ptr. This pointer will point will point to the next address that can be used for allocation as long as the read_ptr does not equal the stop_ptr and the IFC is at its maximum count.

5.3.5 *De-allocate Block*

This block will maintain a free physical address block count for each context. While in current context, a count shall be maintained specifying how many blocks were written into the free list at the write_ptr pointer. This count will be reset upon reset or when this context is active on the back and different than the previous context. It is actually a count of blocks in the previous context that will no longer be used. This count will be used to advance the write_ptr pointer to make available the set of physical blocks freed when the previous context was done. This allows the discard or de-allocation of any number of blocks in one clock.

5.3.6 *Operation of Incremental model*

The basic operation of the model would start with the write_ptr, stop_ptr, read_ptr pointers in the free list set to zero and the free list counter is set to zero. Also all the dirty bits and the previous context will be initialized to zero. When the first set constants happen, the reset dirty bit will not be set, so we will allocate a physical location from the free list counter because its not at the max value. The data will be written into physical address zero. Both the additional copy of the renaming table and the context zeros of the big renaming table will be updated for the logical address that was written by set start with physical address of 0. This process will be repeated for any logical address that are not dirty until the context changes. If a logical address is hit that has its dirty bits set while in the same context, both dirty bits would be set, so the new data will be over-written to the last physical address assigned for this logical address. When the first draw command of the context is detected, the previous context stored in the additional renaming table will be copied to the larger renaming table in the current (new) context location. Then the set constant logical address with be loaded with a new physical address during the copy and if the reset dirty was set, the physical address it replaced in the renaming table would be entered at the write_ptr pointer location on the free list and the write_ptr will be incremented. The de-allocation counter for the previous context (eight) will be incremented. This as set states come in for this context one of the following will happen:

1.) No dirty bits are set for the logical address being updated. A line will be allocated of the free-list counter or the free list at read_ptr pointer if read_ptr != to stop_ptr .
2.) Reset dirty set and Context dirty not set. A new physical address is allocated, the physical address in the renaming table is put on the free list at write_ptr and it is incremented along with the de-allocate counter for the last context.
3.) Context dirty is set then the data will be written into the physical address specified by the logical address.

This process will continue as long as set states arrive. This block will provide backpressure to the CP whenever he has not free list entries available (counter at max and stop_ptr == read_ptr). The command stream will keep a count of contexts of constants in use and prevent more than max constants contexts from being sent.

Whenever a draw packet arrives, the content of the re-mapping table is written to the correct re-mapping table for the context number. Also if the next context uses less constants than the current one all exceeding lines are moved to the free list to be de-allocated later. This happens in parallel with the writing of the re-mapping table to the correct memory.

Now preferable when the constant context leaves the last ALU clause it will be sent to this block and compared with the previous context that left. (Init to zero) If they differ than the older context will no longer be referenced and thus can be de-allocated in the physical memory. This is accomplished by adding the number of blocks freed this context to the stop_ptr pointer. This will make all the physical addresses used by this context available to the read_ptr allocate pointer for future allocation.

This device allows representation of multiple contexts of constants data with N copies of the logical address space. It also allows the second context to be represented as the first set plus some new additional data by just storing the delta's. It allows memory to be efficiently used and when the constants updates are small it can store multiple context. However, if the updates are large, less contexts will be stored and potentially performance will be degraded. Although it will still perform as well as a ring could in this case.

5.4 Constant Store Indexing

In order to do constant store indexing, the sequencer must be loaded first with the indexes (that come from the GPRs). There are 144 wires from the exit of the SP to the sequencer (9 bits pointers x 16 vertexes/clock).

~~Since the data must pass thru the Shader pipe for the float to fixed conversion, there is a latency of 4 clocks (1 instruction) between the time the sequencer is loaded and the time one can index into the constant store. The assembly will look like this~~

MOVA R1.X,R2.X // Loads the sequencer with the content of R2.X, also copies the content of R2.X into R1.X
~~NOP // latency of the float to fixed conversion~~
ADD R3,R4,C0[R2.X]// Uses the state from the sequencer to add R4 to C0[R2.X] into R3

Note that we don't really care about what is in the brackets because we use the state from the MOVA instruction. R2.X is just written again for the sake of simplicity and coherency.

The storage needed in the sequencer in order to support this feature is 2*64*9 bits = 1152 bits.

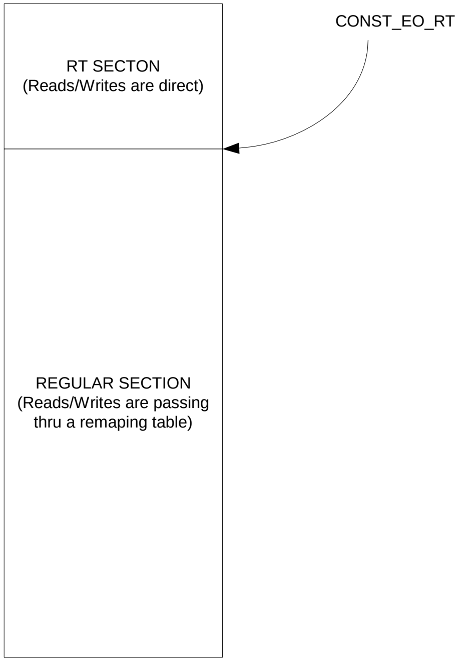
The address register is a signed integer, which ranges from –256 to 255.

5.5 Real Time Commands

The real time commands constants are written by the CP using the register mapped registers allocated for RT. It works is the same way than when dealing with regular constant loads BUT in this case the CP is not sending a logical address but rather a physical address and the reads are not passing thru the re-mapping table but are directly read from the memory. The boundary between the two zones is defined by the CONST_EO_RT control register. Similarly, for the fetch state, the boundary between the two zones is defined by the TSTATE_EO_RT control register.

5.6 Constant Waterfalling

In order to have a reasonable performance in the case of constant store indexing using the address register, we are going to have the possibility of using the physical memory port for read only. This way we can read 1 constant per clock and thus have a worst-case waterfall mode of 1 vertex per clock. There is a small synchronization issue related with this as we need for the SQ to make sure that the constants where actually written to memory (not only sent to the sequencer) before it can allow the first vector of pixels or vertices of the state to go thru the ALUs. To do so, the sequencer keeps 8 bits (one per render state) and sets the bits whenever the last render state is written to memory and clears the bit whenever a state is freed.



CONST_EO_RT
RT SECTON
(Reads/Writes are direct)
REGULAR SECTION
(Reads/Writes are passing thru a remaping table)

Figure 9: The Constant store

6. Looping and Branches

Loops and branches are planned to be supported and will have to be dealt with at the sequencer level. We plan on supporting constant loops and branches using a control program.

6.1 The controlling state.

The R400 controling state consists of:

Boolean[255~~6~~:0]
Loop_count[7:0][31:0]
Loop_Start[7:0][31:0]
Loop_Step[7:0][31:0]

That is 256 Booleans and 32 loops.

We have a stack of 4 elements for nested calls of subroutines and 4 loop counters to allow for nested loops.

This state is available on a per shader program basis.

6.2 The Control Flow Program

We'd like to be able to code up a program of the form:

```
1:	Loop
2:	Exec	TexFetch
```

```
3:              TexFetch
4:              ALU
5:              ALU
6:              TexFetch
7:       End Loop
8:       ALU Export
```

But realize that 3: may be dependent on 2: and 4: is almost certainly dependent on 2: and 3:. Without clausing, these dependencies need to be expressed in the Control Flow instructions. Additionally, without separate 'texture clauses' and 'ALU clauses' we need to know which instructions to dispatch to the Texture Unit and which to the ALU unit. This information will be encapsulated in the flow control instructions.

Each control flow instruction will contain 2 bits of information for each (non-control flow) instruction:

a) ALU or Texture
b) Serialize Execution

(b) would force the thread to stop execution at this point (before the instruction is executed) and wait until all textures have been fetched. Given the allocation of reserved bits, this would mean that the count of an 'Exec' instruction would be limited to about 8 (non-control-flow) instructions. If more than this were needed, a second Exec (with the same conditions) would be issued.

Another function that relies upon 'clauses' is allocation and order of execution. We need to assure that pixels and vertices are exported in the correct order (even if not all execution is ordered) and that space in the output buffers are allocated in order. Additionally data can't be exported until space is allocated. A new control flow instruction:

Alloc <buffer select -- position,parameter, pixel or vertex memory. And the size required>.

would be created to mark where such allocation needs to be done. To assure allocation is done in order, the actual allocation for a given thread can not be performed unless the equivalent allocation for all previous threads is already completed. The implementation would also assure that execution of instruction(s) following the serialization due to the Alloc will occur in order -- at least until the next serialization or change from ALU to Texture. In most cases this will allow the exports to occur without any further synchronization. Only 'final' allocations or position allocations are guaranteed to be ordered. Because strict ordering is required for pixels, parameters and positions, this implies only a single alloc for these structures. Vertex exports to memory do not require ordering during allocation and so multiple 'allocs' may be done.

6.2.1 *Control flow instructions table*

Here is the revised control flow instruction set.

Note that whenever a field is marked as RESERVED, it is assumed that all the bits of the field are cleared (0).

NOP		
47 … 44	43	42 … 0
0000	Addressing	RESERVED

This is a regular NOP.

Execute					
47 … 44	43	40 … 34	33 …16	15…12	11 … 0
0001	Addressing	RESERVED	Instructions type + serialize (9 instructions)	Count	Exec Address

Execute_End					
47 … 44	43	40 … 34	33 …16	15…12	11 … 0
0010	Addressing	RESERVED	Instructions type + serialize (9 instructions)	Count	Exec Address

Execute up to 9 instructions at the specified address in the instruction memory. The Instruction type field tells the sequencer the type of the instruction (LSB) (1 = Texture, 0 = ALU and whether to serialize or not the execution (MSB) (1 = Serialize, 0 = Non-Serialized). If Execute_End this is the last execution block of the shader program.

Conditional_Execute						
47 … 44	43	42	41 … 34	33…16	15 …12	11 … 0
0011	Addressing	Condition	Boolean address	Instructions type + serialize (9 instructions)	Count	Exec Address

Conditional_Execute_End						
47 … 44	43	42	41 … 34	33…16	15 …12	11 … 0
0100	Addressing	Condition	Boolean address	Instructions type + serialize (9 instructions)	Count	Exec Address

If the specified Boolean (8 bits can address 256 Booleans) meets the specified condition then execute the specified instructions (up to 9 instructions). If the condition is not met, we go on to the next control flow instruction. If Conditional_Execute_End and the condition is met, this is the last execution block of the shader program.

Conditional_Execute_Predicates							
47 … 44	43	42	41 … 36	35 … 34	33…16	15…12	11 … 0
0101	Addressing	Condition	RESERVED	Predicate vector	Instructions type + serialize (9 instructions)	Count	Exec Address

Conditional_Execute_Predicates_End							
47 … 44	43	42	41 … 36	35 … 34	33…16	15…12	11 … 0
0110	Addressing	Condition	RESERVED	Predicate vector	Instructions type + serialize (9 instructions)	Count	Exec Address

Check the AND/OR of all current predicate bits. If AND/OR matches the condition execute the specified number of instructions. We need to AND/OR this with the kill mask in order not to consider the pixels that aren't valid. If the condition is not met, we go on to the next control flow instruction. If Conditional_Execute_Predicates_End and the condition is met, this is the last execution block of the shader program.

Conditional_Execute_Predicates_No_Stall							
47 … 44	43	42	41 … 36	35 … 34	33…16	15…12	11 … 0
1101	Addressing	Condition	RESERVED	Predicate vector	Instructions type + serialize (9 instructions)	Count	Exec Address

Conditional_Execute_Predicates_No_Stall_End							
47 … 44	43	42	41 … 36	35 … 34	33…16	15…12	11 … 0
1110	Addressing	Condition	RESERVED	Predicate vector	Instructions type + serialize (9 instructions)	Count	Exec Address

Same as Conditionnal_Execute_Predicates but the SQ is not going to wait for the predicate vector to be updated. You can only set this in the compiler if you know that the predicate set is only a refinement of the current one (like a nested if) because the optimization would still work.

Loop_Start					
47 … 44	43	42 … 21	20 … 16	15…13	12 … 0
0111	Addressing	RESERVED	loop ID	RESERVED	Jump address

Loop Start. Compares the loop iterator with the end value. If loop condition not met jump to the address. Forward jump only. Also computes the index value. The loop id must match between the start to end, and also indicates which control flow constants should be used with the loop.

Loop_End									
47 … 44	43	42 ~~… 24~~	41… 36	35…34	33… 22~~23… 21~~	21	20 … 16	15…13	12 … 0
1000	Addressing	~~RESERVED~~Cond	RESERVED	Predicate Vector	RESERVED ~~Predicate break~~	Pred break	loop ID	RESERVED	start address

Loop end. Increments the counter by one, compares the loop count with the end value. If loop condition met, continue, else, jump BACK to the start of the loop. If predicate break != 0, then compares predicate vector n (specified by predicate ~~break number~~Vector) to condition. If all bits ~~cleared then break the loop~~meet condition then break the loop.

The way this is described does not prevent nested loops, and the inclusion of the loop id make this easy to do.

Conditionnal_Call						
47 … 44	43	42	41 … 34	33 … 14	13	12 … 0
1001	Addressing	Condition	Boolean address	RESERVED	Force Call	Jump address

If the condition is met, jumps to the specified address and pushes the control flow program counter on the stack. If force call is set the condition is ignored and the call is made always.

Return		
47 … 44	43	42 … 0
1010	Addressing	RESERVED

Pops the topmost address from the stack and jumps to that address. If nothing is on the stack, the program will just continue to the next instruction.

Conditionnal_Jump							
47 … 44	43	42	41… 34	33	32 … 14	13	12 … 0
1011	Addressing	Condition	Boolean address	FW only	RESERVED	Force Jump	Jump address

If force jump is set the condition is ignored and the jump is made always. If FW only is set then only forward jumps are allowed.

Allocate				
47 … 44	43	42…41	40 … 3	2…0
1100	Debug	Buffer Select	RESERVED	Size

Buffer Select takes a value of the following:
01 – position export (ordered export)
10 – parameter cache or pixel export (ordered export)
11 – pass thru (out of order exports).

Size field is only used to reserve space in the export buffer for pass thru exports. Valid values are 1 (1 line) thru 9 (9 lines). It should be determined by the compiler/assembler by taking max index used +1.

If debug is set this is a debug alloc (ignore if debug DB_ON register is set to off).

6.3 Implementation

The envisioned implementation has a buffer that maintains the state of each thread. A thread lives in a given location in the buffer during its entire life, but the buffer has FIFO qualities in that threads leave in the order that they enter. Actually two buffers are maintained -- one for Vertices and one for Pixels. The intended implementation would allow for:

16 entries for vertices
48 entries for pixels.

From each buffer, arbitration logic attempts to select 1 thread for the texture unit and ~~1~~ 2 (interleaved) thread for the ALU unit. Once a thread is selected it is read out of the buffer, marked as invalid, and submitted to appropriate execution unit. It is returned to the buffer (at the same place) with its status updated once all possible sequential instructions have been executed. A switch from ALU to TEX or visa-versa or a Serialize_Execution modifier forces the thread to be returned to the buffer.

Each entry in the buffer will be stored across two physical pieces of memory - most bits will be stored in a 1 read port device. Only bits needed for thread arbitration will be stored in a highly multi-ported structure. The bits kept in the 1 read port device will be termed 'state'. The bits kept in the multi-read ported device will be termed 'status'.

'State Bits' needed include:

1. Control Flow Instruction Pointer (13 bits),
2. Execution Count Marker 4 bits),
3. Loop Iterators (4x9 bits),
4. Loop Counters (4x9 bits),
5. Call return pointers (4x13 bits),
6. Predicate Bits (64 bits),
7. Export ID (1 bit),
8. Parameter Cache base Ptr (7 bits),
9. GPR Base Ptr (8 bits),
10. Context Ptr (3 bits).
11. LOD corrections (6x16 bits)
12. Valid bits (64 bits)
13. RT (1 bit) Signifies that this thread is a Real Time thread. This bit must be sent to the Constant store state machine when reading it.

Absent from this list are 'Index' pointers. These are costly enough that I'm presuming that they are instead stored in the GPRs. The first seven fields above (Control Flow Ptr, Execution Count, Loop Counts, call return ptrs, Predicate bits, PC base ptr and export ID) are updated every time the thread is returned to the buffer based on how much progress has been mode on thread execution. GPR Base Ptr, Context Ptr and LOD corrections are unchanged throughout execution of the thread.

'Status Bits' needed include:

- Valid Thread
- Texture/ALU engine needed
- Texture Reads are outstanding
- Waiting on Texture Read to Complete
- Allocation Wait (2 bits)
- 00 – No allocation needed
- 01 – Position export allocation needed (ordered export)
- 10 – Parameter or pixel export needed (ordered export)
- 11 – pass thru (out of order export)
- Allocation Size (4 bits)
- Position Allocated
- Mem/Color Allocated
- First thread of a new context
- Event thread (NULL thread that needs to trickle down the pipe)

- Last (1 bit)
- Pulse SX (1 bit)

All of the above fields from all of the entries go into the arbitration circuitry. The arbitration circuitry will select a winner for both the Texture Engine and for the ALU engine. There are actually two sets of arbitration -- one for pixels and one for vertices. A final selection is then done between the two. But the rest of this implementation summary only considers the 'first' level selection which is similar for both pixels and vertices.

Texture arbitration requires no allocation or ordering so it is purely based on selecting the 'oldest' thread that requires the Texture Engine.

ALU arbitration is a little more complicated. First, only threads where either of Texture_Reads_outstanding or Waiting_on_Texture_Read_to_Complete are '0' are considered. Then if Allocation_Wait is active, these threads are further filtered based on whether space is available. If the allocation is position allocation, then the thread is only considered if all 'older' threads have already done their position allocation (position allocated bits set). If the allocation is parameter or pixel allocation, then the thread is only considered if it is the oldest thread. Also a thread is not considered if it is a parameter or pixel or position allocation, has its First_thread_of_a_new_context bit set and would cause ALU interleaving with another thread performing the same parameter or pixel or position allocation. Finally the 'oldest' of the threads that pass through the above filters is selected. If the thread needed to allocate, then at this time the allocation is done, based on Allocation_Size. If a thread has its “last” bit set, then it is also removed from the buffer, never to return.

If I now redefine 'clauses' to mean 'how many times the thread is removed from the thread buffer for the purpose of exection by either the ALU or Texture engine', then the minimum number of clauses needed is 2 -- one to perform the allocation for exports (execution automatically halts after an 'Alloc' instruction) (but doesn't performs the actual allocation) and one for the actual ALU/export instructions. As the 'Alloc' instruction could be part of a texture clause (presumably the final instruction in such a clause), a thread could still execute in this minimal number of 2 clauses, even if it involved texture fetching.

The Texture_Reads_Outstanding bit must be updated by the sequencer, based on keeping track of how many Texture Clauses have been executed by a given thread that have not yet had there data returned. Any number above 0 results in this bit being set. We could consider forcing synchronization such that two texture clauses for a given thread may not be outstanding at any time (that would be my preference for simplicity reasons and because it would require only very little change in the texture pipe interface). This would allow the sequencer to set the bit on execution of the texture clause, and allow the texture unit to return a pointer to the thread buffer on completion that clears the bit.

6.4 Data dependant predicate instructions

Data dependant conditionals will be supported in the R400. The only way we plan to support those is by supporting three vector/scalar predicate operations of the form:

PRED_SETE~~_#~~ _PUSH - similar to SETE except that the result is 'exported' to the sequencer.
PRED_SETNE_PUSH# - similar to SETNE except that the result is 'exported' to the sequencer.
PRED_SETGT_PUSH ~~#~~- similar to SETGT except that the result is 'exported' to the sequencer
PRED_SETGTE_PUSH# - similar to SETGTE except that the result is 'exported' to the sequencer

For the scalar operations only we will also support the two following instructions:
PRED_SETE~~0_# – SETE0~~

PRED_SET~~E1_#~~NE ~~– SETE~~
PRED_SETGT ~~1~~
PRED_SET_INV
PRED_SET_POP
PRED_SET_CLR
PRED_SET_RESTORE

Details about actual implementation of these opcodes are in the shader pipe architectural spec.

The export is a single bit - 1 or 0 that is sent using the same data path as the MOVA instruction. The sequencer will maintain ~~4~~ 1 set~~s~~ of 64 bits predicate vectors (in fact 2~~8~~ sets because we interleave two programs but only ~~4~~ 1 will be exposed) and use it to control the write masking. This predicate is ~~not~~ maintained across clause boundaries. ~~The # sign is used to specify which predicate set you want to use 0 thru 3.~~

Then we have two conditional execute bits. The first bit is a conditional execute “on” bit and the second bit tells us if we execute on 1 or 0. For example, the instruction:

~~P0~~_P0_ ADD_# R0,R1,R2

Is only going to write the result of the ADD into those GPRs whose predicate bit is 0. Alternatively, P1_ADD_# would only write the results to the GPRs whose predicate bit is set. The use of the P0 or P1 without precharging the sequencer with a PRED instruction is undefined.

~~{Issue: do we have to have a NOP between PRED and the first instruction that uses a predicate?}~~

6.5 HW Detection of PV,PS

Because of the control program, the compiler cannot detect statically dependant instructions. In the case of non-masked writes and subsequent reads the sequencer will insert uses of PV,PS as needed. This will be done by comparing the read address and the write address of consecutive instructions. For masked writes, the sequencer will insert ~~NOPs wherever there is a dependant read/write.~~detect wich channels to read from the GPRs and which ones to read from the PV/PS.

~~The sequencer will also have to insert NOPs between PRED_SET and MOVA instructions and their uses.~~

6.6 Register file indexing

Because we can have loops in fetch clause, we need to be able to index into the register file in order to retrieve the data created in a fetch clause loop and use it into an ALU clause. The instruction will include the base address for register indexing and the instruction will contain these controls:

Bit7	Bit 6	
0	0	'absolute register'
0	1	'relative register'
1	0	'previous vector'
1	1	'previous scalar'

In the case of an absolute register we just take the address as is. In the case of a relative register read we take the base address and we add to it the loop_index and this becomes our new address that we give to the shader pipe.

The sequencer is going to keep a loop index computed as such:

Index = Loop_iterator*Loop_step + Loop_start.

We loop until loop_iterator = loop_count. Loop_step is a signed value [-128…127]. The computed index value is a 10 bit counter that is also signed. Its real range is [-256,256]. The tenth bit is only there so that we can provide an out of range value to the “indexing logic” so that it knows when the provided index is out of range and thus can make the necessary arrangements.

6.7 Debugging the Shaders

In order to be able to debug the pixel/vertex shaders efficiently, we provide 2 methods.

6.7.1 *Method 1: Debugging registers*

Current plans are to expose 2 debugging, or error notification, registers:
1. address register where the first error occurred
2. count of the number of errors

The sequencer will detect the following groups of errors:
- count overflow
- constant indexing overflow
- register indexing overflow

Compiler recognizable errors:
- jump errors
  relative jump address > size of the control flow program
- call stack
  call with stack full
  return with stack empty

~~A jump error will always cause the program to break. In this case, a break means that a clause will halt execution, but allowing further clauses to be executed.~~

With all the other errors, program can continue to run, potentially to worst-case limits. ~~The program will only break if the DB_PROB_BREAK register is set.~~

If indexing outside of the constant or the register range, causing an overflow error, the hardware is specified to return the value with an index of 0. This could be exploited to generate error tokens, by reserving and initializing the 0th register (or constant) for errors.

{ISSUE : Interrupt to the driver or not?}

6.7.2 *Method 2: Exporting the values in the GPRs*

1) The sequencer will have a debug active, count register and an address register for this mode.

Under the normal mode execution follows the normal course.

Under the debug mode it is assumed that the program is always exporting n debug vectors and that all other exports to the SX block (but for position~~, color, z, ect~~) will been~~en~~ turned off (changed into NOPs) by the sequencer (even if they occur before the address stated by the ADDR debug register).

7. Pixel Kill Mask

A vector of 64 bits is kept by the sequencer per group of pixels/vertices. Its purpose is to optimize the texture fetch requests and allow the shader pipe to kill pixels using the following instructions:

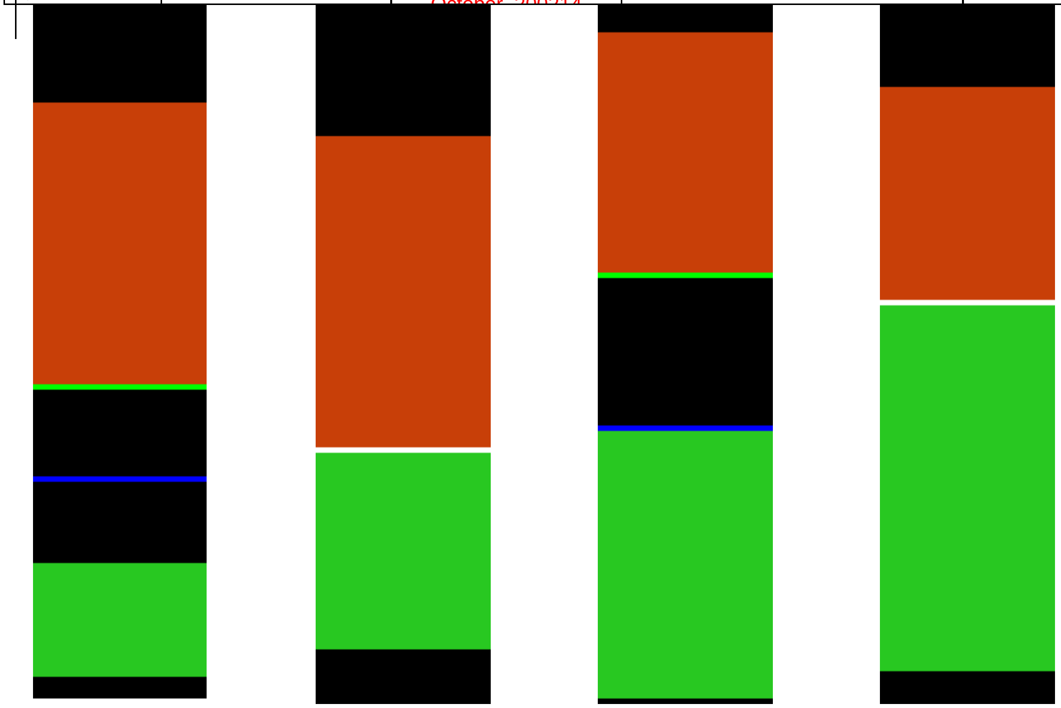
MASK_SETE
MASK_SETNE
MASK_SETGT
MASK_SETGTE

8. Multipass vertex shaders (HOS)

Multipass vertex shaders are able to export from the 6 last clauses but to memory ONLY.

9. Register file allocation

The register file allocation for vertices and pixels can either be static or dynamic. In both cases, the register file in managed using two round robins (one for pixels and one for vertices). In the dynamic case the boundary between pixels and vertices is allowed to move, in the static case it is fixed to 128-VERTEX_REG_SIZE for vertices and PIXEL_REG_SIZE for pixels.



Above is an example of how the algorithm works. Vertices come in from top to bottom; pixels come in from bottom to top. Vertices are in orange and pixels in green. The blue line is the tail of the vertices and the green line is the tail of the pixels. Thus anything between the two lines is shared. When pixels meets vertices the line turns white and the boundary is static until both vertices and pixels share the same “unallocated bubble”. Then the boundary is allowed to move again. The numbering of the GPRs starts from the bottom of the picture at index 0 and goes up to the top at index 127.

10. Fetch Arbitration

The fetch arbitration logic chooses one of the n potentially pending fetch clauses to be executed. The choice is made by looking at the Vs and Ps reservation stations and picking the first one ready to execute. Once chosen, the clause state machine will send one 2x2 fetch per clock (or 4 fetches in one clock every 4 clocks) until all the fetch instructions of the clause are sent. This means that there cannot be any dependencies between two fetches of the same clause.

The arbitrator will not wait for the fetches to return prior to selecting another clause for execution. The fetch pipe will be able to handle up to X(?) in flight fetches and thus there can be a fair number of active clauses waiting for their fetch return data.

11. ALU Arbitration

ALU arbitration proceeds in almost the same way than fetch arbitration. The ALU arbitration logic chooses one of the n potentially pending ALU clauses to be executed. The choice is made by looking at the Vs and Ps reservation stations and picking the first one ready to execute. There are two ALU arbiters, one for the even clocks and one for the odd clocks. For example, here is the sequencing of two interleaved ALU clauses (E and O stands for Even and Odd sets of 4 clocks):

Einst0 Oinst0 Einst1 Oinst1 Einst2 Oinst2 Einst0 Oinst3 Einst1 Oinst4 Einst2 Oinst0…

Proceeding this way hides the latency of 8 clocks of the ALUs. Also note that the interleaving also occurs across clause boundaries.

12. Handling Stalls

When the output file is full, the sequencer prevents the ALU arbitration logic from selecting the last clause (this way nothing can exit the shader pipe until there is place in the output file. If the packet is a vertex packet and the position buffer is full (POS_FULL) then the sequencer also prevents a thread from entering an exporting clause. The sequencer will set the OUT_FILE_FULL signal n clocks before the output file is actually full and thus the ALU arbiter will be able read this signal and act accordingly by not preventing exporting clauses to proceed.

13. Content of the reservation station FIFOs

The reservation FIFOs contain the state of the vector of pixels and vertices. We have two sets of those: one for pixels, and one for vertices. They contain 3 bits of Render State 7 bits for the base address of the GPRs, some bits for LOD correction and coverage mask information in order to fetch fetch for only valid pixels, the quad address.

14. The Output File

The output file is where pixels are put before they go to the RBs. The write BW to this store is 256 bits/clock. Just before this output file are staging registers with write BW 512 bits/clock and read BW 256 bits/clock. The staging registers are 4x128 (and there are 16 of those on the whole chip).

15. IJ Format

The IJ information sent by the PA is of this format on a per quad basis:

We have a vector of IJ's (one IJ per pixel at the centroid of the fragment or at the center of the pixel depending on the mode bit). All pixel's parameters are always interpolated at full 20x24 mantissa precision.

$$P0 = A + I(0)*(B - A) + J(0)*(C - A)$$
$$P1 = A + I(1)*(B - A) + J(1)*(C - A)$$
$$P2 = A + I(2)*(B - A) + J(2)*(C - A)$$
$$P3 = A + I(3)*(B - A) + J(3)*(C - A)$$

P0	P1
P2	P3

Multiplies (Full Precision): 8
Subtracts 19x24 (Parameters): 2
Adds: 8

FORMAT OF P's IJ : Mantissa 20 Exp 4 for I + Sign
Mantissa 20 Exp 4 for J + Sign

Total number of bits : 20*8 + 4*8 + 4*2 = 200.

All numbers are kept using the un-normalized floating point convention: if exponent is different than 0 the number is normalized if not, then the number is un-normalized. The maximum range for the IJs (Full precision) is +/- 1024.

15.1 Interpolation of constant attributes

Because of the floating point imprecision, we need to take special provisions if all the interpolated terms are the same or if two of the terms are the same.

16. Staging Registers

In order for the reuse of the vertices to be 14, the sequencer will have to re-order the data sent IN ORDER by the VGT for it to be aligned with the parameter cache memory arrangement. Given the following group of vertices sent by the VGT:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 || 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 || 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 || 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63

The sequencer will re-arrange them in this fashion:

0 1 2 3 16 17 18 19 32 33 34 35 48 49 50 51 || 4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55 || 8 9 10 11 24 25 26 27 40 41 42 43 56 57 58 59 || 12 13 14 15 28 29 30 31 44 45 46 47 60 61 62 63

The || markers show the SP divisions. In the event a shader pipe is broken, the SQ is responsible to insert padding to account for the missing pipe. For example, if SP1 is broken, vertices 4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55 will not be sent by the VGT to the SQ **AND** the SQ is responsible to “jump” over these vertices in order for no valid vertices to be sent to an invalid SP.

The most straightforward, *non-compressed* interface method would be to convert, in the VGT, the data to 32-bit floating point prior to transmission to the VSISRs. In this scenario, the data would be transmitted to (and stored in) the VSISRs in full 32-bit floating point. This method requires three 24-bit fixed-to-float converters in the VGT. Unfortunately, it also requires and additional 3,072 bits of storage across the VSISRs. This interface is illustrated in Figure 11~~Figure 11Figure 11~~. The area of the fixed-to-float converters and the VSISRs for this method is roughly estimated as 0.759sqmm using the R300 process. The gate count estimate is shown in Figure 10~~Figure 10Figure 10~~.

Basis for 8-deep Latch Memory (from R300)			
8x24-bit	11631 μ^2		60.57813 μ^2 per bit
Area of 96x8-deep Latch Memory	46524 μ^2		
Area of 24-bit Fix-to-float Converter	4712 μ^2 per converter		
Method 1	Block	Quantity	Area
	F2F	3	14136
	8x96 Latch	16	744384
			758520 μ^2

Figure 10:Area Estimate for VGT to Shader Interface

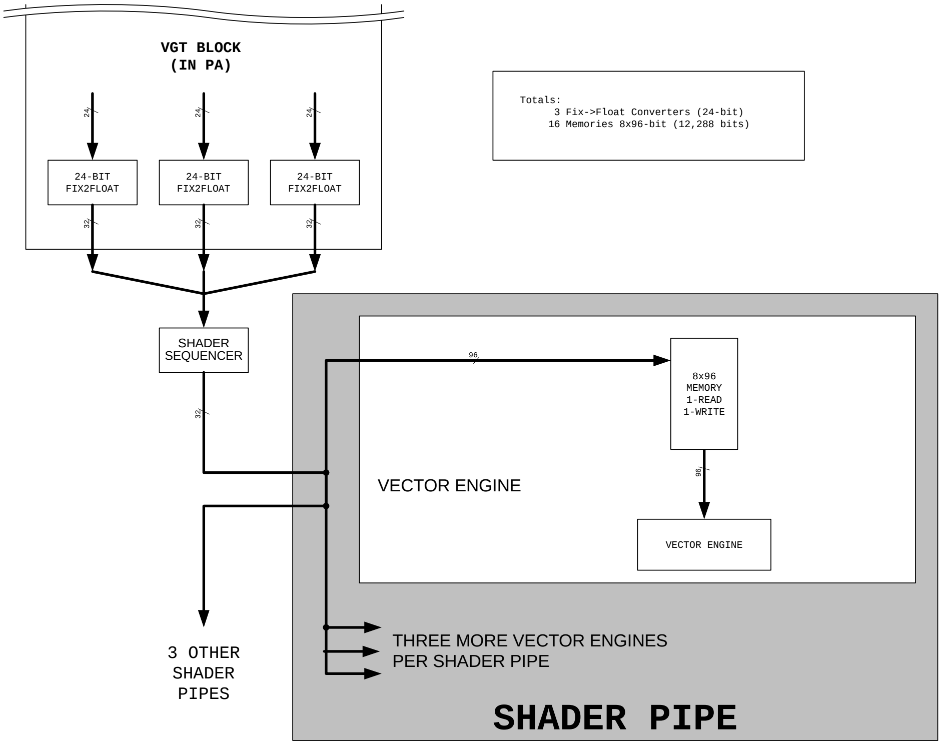


Figure 11:VGT to Shader Interface

17. The parameter cache

The parameter cache is where the vertex shaders export their data. It consists of 16 128x128 memories (1R/1W). The reuse engine will make it so that all vertexes of a given primitive will hit different memories. The allocation method for these memories is a simple round robin. The parameter cache pointers are mapped in the following way: 4MSBs are the memory number and the 7 LSBs are the address within this memory.

MEMORY NUMBER 4 bits	ADDRESS 7 bits

The PA generates the parameter cache addresses as the positions come from the SQ. All it needs to do is keep a Current_Location pointer (7 bits only) and as the positions comes increment the memory number. When the memory number field wraps around, the PA increments the Current_Location by VS_EXPORT_COUNT (a snooped register from the SQ). As an example, say the memories are all empty to begin with and the vertex shader is exporting 8 parameters per vertex (VS_EXPORT_COUNT = 8). The first position received is going to have the PC address 00000000000 the second one 00010000000, third one 00100000000 and so on up to 11110000000. Then the next position received (the 17th) is going to have the address 00000001000, the 18th 00010001000, the 19th 00100001000 and so on. The Current_location is NEVER reset BUT on chip resets. The only thing to be careful about is that if the SX doesn't send you a full group of positions (<64) then you need to fill the address space so that the next group starts correctly aligned (for example if you receive only 33 positions then you need to add 2*VS_EXPORT_COUNT to Current_Location and reset the memory count to 0 before the next vector begins).

17.1 Export restrictions

17.1.1 *Pixel exports:*

Pixels can export 1,2,3 or 4 color buffers to the SX(+z). The exports will be done in order. ~~The PRED_OPTIMIZE function has to be turned of if the exports are done using interleaved predicated instructions.~~ The exports will always be ordered to the SX.

17.1.2 *Vertex exports:*

Position or parameter caches can be exported in any order in the shader program. It is always better to export posistion as soon as possible. Position has to be exported in a single export block (no texture instructions can be placed between the exports). Parameter cache exports can be done in any order with texture instructions interleaved. ~~The PRED_OPTIMIZE function has to be turned of if the exports are done using interleaved predicated instructions to the Parameter cache (see Arbitration restrictions for details).~~ The exports will always be allocated in order to the SX.

17.1.3 *Pass thru exports:*

Pass thru exports have to be done in groups of the form:

```
Alloc 4 (or 8~~8 or 12~~)
Execute ALU(ADDR) ALU(DATA) ALU(DATA) ALU(DATA)…
```

They cannot have texture instructions interleaved in the export block. These exports **are not guaranteed to be ordered**.

Also, when doing a pass thru export**, Position MUST be exported AFTER all pass thru exports**. This position export is used to synchronize the chip when doing a transition from pass thru shader to regular shader and vice versa.

Formatted

17.2 Arbitration restrictions

Here are the Sequencer arbitration restrictions:

1) Cannot execute a serialized thread if the corresponding texture pending bit is set
2) Cannot allocate position if any older thread has not allocated position
3) Cannot have more than 2 opened allocs of type : Memory, position and Color.
~~3)~~4) If last thread is marked as not valid AND marked as last and we are about to execute the second to oldest thread also marked last then:
   a. Both threads must be from the same context (cannot allow a first thread)
   b. Must turn off the predicate optimization for the second thread
~~4)~~5) Cannot execute a texture clause if texture reads are pending
~~5)~~6) Cannot execute last if texture pending (even if not serial)
7) Cannot allocate if not last or second to last for color exports.

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

18. Export Types

The export type (or the location where the data should be put) is specified using the destination address field in the ALU instruction. Here is a list of all possible export modes:

18.1 Vertex Shading

0:15	- 16 parameter cache
16:31	- Empty (Reserved?)
32	- Export Address
33:37	- 5 vertex exports to the frame buffer and index
38:47	- Empty
48:52	- 5 debug export (interpret as normal memory export)

60	- export addressing mode
61	- Empty
62	- position
63	- sprite size export that goes with position export

(X= point size, Y= edge flag is bit 0, Z= VtxKill is bitwise OR of bits 30:0. Any bit other than sign means VtxKill.)

18.2 Pixel Shading

0	- Color for buffer 0 (primary)
1	- Color for buffer 1
2	- Color for buffer 2
3	- Color for buffer 3
4:15	- Empty
16	- Buffer 0 Color/Fog (primary)
17	- Buffer 1 Color/Fog
18	- Buffer 2 Color/Fog
19	- Buffer 3 Color/Fog
20:31	- Empty
32	- Export Address
33:37	- 5 exports for multipass pixel shaders.
38:47	- Empty
48:52	- 5 debug exports (interpret as normal memory export)
60	- export addressing mode
61	- Z for primary buffer (Z exported to 'alpha' component)
62:63	- Empty

19. Special Interpolation modes

19.1 Real time commands

We are unable to use the parameter memory since there is no way for a command stream to write into it. Instead we need to add three 16x128 memories (one for each of three vertices x 16 interpolants). These will be mapped onto the register bus and written by type 0 packets, and output to the the parameter busses (the sequencer and/or PA need to be able to address the realtime parameter memory as well as the regular parameter store. For higher performance we should be able able to view them as two banks of 16 and do double buffering allowing one to be loaded, while the other is rasterized with. Most overlay shaders will need 2 or 4 scalar coordinates, one option might be to restrict the memory to 16x64 or 32x64 allowing only two interpolated scalars per cycle, the only problem I see with this is, if we view support for 16 vector-4 interpolants important (true only if we map Microsoft's high priority stream to the realtime stream), then the PA/sequencer need to support a realtime-specific mode where we need to address 32 vectors of parameters instead of 16. This mode is triggered by the primitive type: REAL TIME. The actual memories are in the in the SX blocks. The parameter data memories are hooked on the RBBM bus and are loaded by the CP using register mapped memory.

19.2 Sprites/ XY screen coordinates/ FB information

XY screen coordinates may be needed in the shader program. This functionality is controlled by the param_gen_I0 register (in SQ) in conjunction with the SND_XY register (in SC) and the param_gen_pos. Also it is possible to send the faceness information (for OGL front/back special operations) to the shader using the same control register. Here is a list of all the modes and how they interact together:

The Data is going to be written in the register specified by the param_gen_pos register.

Param_Gen_I0 disable, snd_xy disable = No modification
Param_Gen_I0 disable, snd_xy enable = No modification
Param_Gen_I0 enable, snd_xy disable = Sign(faceness)garbage,(Sign Point)garbage,Sign(Line)s, t
Param_Gen_I0 enable, snd_xy enable = Sign(faceness)screenX,(Sign Point)screenY,Sign(Line)s, t

In other words,

The generated vector is (X in RED, Y in GREEN, S in BLUE and T in ALPHA):
X,Y,S,T

These values are always supposed to be positive and any shader use of them should use the ABS function (as their sign bits will now be used for flags).

SignX = BackFacing
SignY = Point Primitive
SignS = Line Primitive
SignT = currently unused as a flag.

If !Point & !Line, then it is a Poly.

I would assume that one implementation which allows for generic texture lookup (using 3D maps) for poly stipple and AA for the driver would be

```
if(Y<0) {
        R = 0.0 (Point)
} else if (S < 0) {
        R = 1.0 (Line)
} else {
        R = 2.0 (Poly)
}
```

19.3 Auto generated counters

In the cases we are dealing with multipass shaders, the sequencer is going to generate a vector count to be able to both use this count to write the 1st pass data to memory and then use the count to retrieve the data on the 2nd pass. The count is always generated in the same way but it is passed to the shader in a slightly different way depending on the shader type (pixel or vertex). This is toggled on and off using the GEN_INDEX_PIX/VTX register. The sequencer is going to keep two counters, one for pixels and one for vertices. Every time a full vector of vertices or pixels is written to the GPRs the counter is incremented. Every time a RST_PIX_COUNT or RST_VTX_COUNT events are received, the corresponding counter is reset. While there is only one count broadcast to the GPRs, the LSB are hardwired to specific values making the index different for all elements in the vector. Since the count must be different for all pixels/vertices and the 4 LSBs (16 positions) are hardwired to the corresponding shader unit the SQ has two choices:

1) Maintain a 19 bit counter that counts the vectors of 64. In this case the phase must be appended to the count before the count is broadcast to the SPs:

Counter (19 bits)	Phase (2 bits)	Hardwired (4 bits)

2) Maintain a 21 bits counter that counts sub-vectors of 16. In this case only the counter is sent to the Sps:

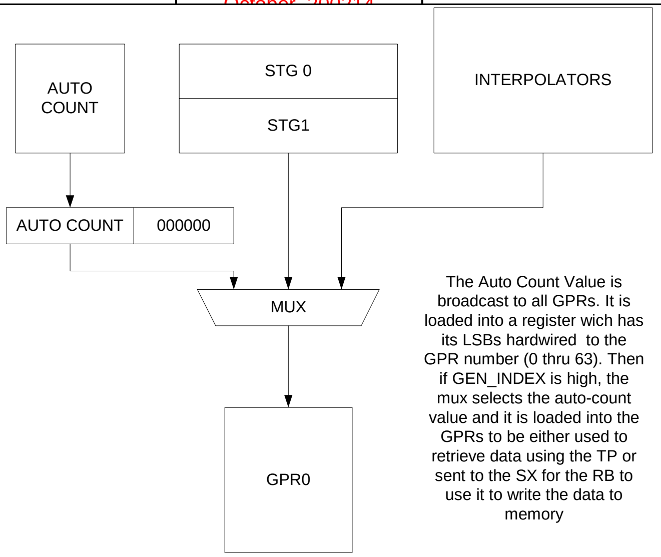
Counter (21 bits)	Hardwired (4 bits)

19.3.1 *Vertex shaders*

In the case of vertex shaders, if GEN_INDEX_VTX is set, the data will be put into the x field of the third register (it means that the compiler must allocate 3 GPRs in all multipass vertex shader modes).

19.3.2 *Pixel shaders*

In the case of pixel shaders, if GEN_INDEX_PIX is set, the data will be put in the x field of the param_gen_pos+1 register.



The Auto Count Value is broadcast to all GPRs. It is loaded into a register wich has its LSBs hardwired to the GPR number (0 thru 63). Then if GEN_INDEX is high, the mux selects the auto-count value and it is loaded into the GPRs to be either used to retrieve data using the TP or sent to the SX for the RB to use it to write the data to memory

Figure 12: GPR input mux Control

20. State management

Every clock, the sequencer will report to the CP the oldest states still in the pipe. These are the states of the programs as they enter the last ALU clause.

20.1 Parameter cache synchronization

In order for the sequencer not to begin a group of pixels before the associated group of vertices has finished, the sequencer will keep a 6 bit count per state (for a total of 8 counters). These counters are initialized to 0 and every time a vertex shader exports its data TO THE PARAMETER CACHE, the corresponding pointer is incremented. When the SC sends a new vector of pixels with the SC_SQ_new_vector bit asserted, the sequencer will first check if the count is greater than 0 before accepting the transmission (it will in fact accept the transmission but then lower its ready to receive). Then the sequencer waits for the count to go to one and decrements it. The sequencer can then issue the group of pixels to the interpolators. Every time the state changes, the new state counter is initialized to 0.

21. XY Address imports

The SC will be able to send the XY addresses to the GPRs. It does so by interleaving the writes of the IJs (to the IJ buffer) with XY writes (to the XY buffer). Then when writing the data to the GPRs, the sequencer is going to interpolate the IJ data or pass the XY data thru a Fix→float converter and expander and write the converted values to the GPRs. The Xys are currently SCREEN SPACE COORDINATES. The values in the XY buffers will wrap. See section 19.2 for details on how to control the interpolation in this mode.

21.1 Vertex indexes imports

In order to import vertex indexes, we have 16 8x96 staging registers. These are loaded one line at a time by the VGT block (96 bits). They are loaded in floating point format and can be transferred in 4 or 8 clocks to the GPRs.

22. Registers

Please see the auto-generated web pages for register definitions.

23. Interfaces

23.1 External Interfaces

Whenever an x is used, it means that the bus is broadcast to all units of the same name. For example, if a bus is named SQ→SPx it means that SQ is going to broadcast the same information to all SP instances.

23.2 SC to SP Interfaces

23.2.1 *SC_SP#*

There is one of these interfaces at front of each of the SP (buffer to stage pixel interpolators). This interface transmits the I,J data for pixel interpolation. For the entire system, two quads per clock are transferred to the 4 SPs, so each of these 4 interfaces transmits one half of a quad per clock. The interface below describes a half of a quad worth of data.

The actual data which is transferred per quad is

Ref Pix I => S4.20 Floating Point I value *4
Ref Pix J => S4.20 Floating Point J value *4

This equates to a total of 200 bits which transferred over 2 clocks
and therefor needs an interface 100 bits wide

Additionally, X,Y data (12-bit unsigned fixed) is conditionally sent across this data bus over the same wires in an additional clock. The X,Y data is sent on the lower 24 bits of the data bus with faceness in the msb.
Transfers across these interfaces are synchronized with the SC_SQ IJ Control Bus transfers.

The data transfer across each of these busses is controlled by a IJ_BUF_INUSE_COUNT in the SC. Each time the SC has sent a pixel vector's worth of data to the SPs, he will increment the IJ_BUF_INUSE_COUNT count. Prior to sending the next pixel vectors data, he will check to make sure the count is less than MAX_BUFER_MINUS_2, if not the SC will stall until the SQ returns a pipelined pulse to decrement the count when he has scheduled a buffer free. Note: We could/may optimize for the case of only sending only IJ to use all the buffers to pre-load more. Currently it is planned for the SP to hold 2 double buffers of I,J data and two buffers of X,Y data, so if either X,Y or Centers and Centroids are on, then the SC can send two Buffers.

In at least the initial version, the SC shall send 16 quads per pixel vector even if the vector is not full. This will increment buffer write address pointers correctly all the time. (We may revisit this for both the SX,SP,SQ and add a EndOfVector signal on all interfaces to quit early. We opted for the simple mode first with a belief that only the end of packet and multiple new vector signals should cause a partial vector and that this would not really be significant performance hit.)

Name	Bits	Description
SC_SP#_data	100	IJ information sent over 2 clocks (or X,Y in 24 LSBs with faceness in upper bit) **Type 0 or 1**, First clock I, second clk J Field ULC URC LLC LRC Bits [63:39] [38:26] [25:13] [12:0] Format SE4M20 SE4M20 SE4M20 SE4M20 **Type 2** Field Face X Y Bits [24] [23:12] [11:0] Format Bit Unsigned Unsigned
SC_SP#_valid	1	Valid
SC_SP#_last_quad_data	1	This bit will be set on the last transfer of data per quad.
SC_SP#_type	2	0 -> Indicates centroids 1 -> Indicates centers 2 -> Indicates X,Y Data and faceness on data bus The SC shall look at state data to determine how many types to send for the

		interpolation process.

The # is included for clarity in the spec and will be replaced with a prefix of u#_ in the verilog module statement for the SC and the SP block will have neither because the instantiation will insert the prefix.

23.2.2 *SC_SQ*

This is the control information sent to the sequencer in order to synchronize and control the interpolation and/or loading data into the GPRs needed to execute a shader program on the sent pixels. This data will be sent over two clocks per transfer with 1 to 16 transfers. Therefore the bus (approx 108 bits) could be folded in half to approx 54 bits.

Name	**Bits**	**Description**
SC_SQ_data	46	Control Data sent to the SQ 1 clk transfers Event – valid data consist of event_id and state_id. Instruct SQ to post an event vector to send state id and event_id through request fifo and onto the reservation stations making sure state id and/or event_id gets back to the CP. Events only follow end of packets so no pixel vectors will be in progress. Empty Quad Mask – Transfer Control data consisting of pc_dealloc or new_vector. Receipt of this is to transfer pc_dealloc or new_vector without any valid quad data. New vector will always be posted to request fifo and pc_dealloc will be attached to any pixel vector outstanding or posted in request fifo if no valid quad outstanding. 2 clk transfers Quad Data Valid – Sending quad data with or without new_vector or pc_dealloc. New vector will be posted to request fifo with or without a pixel vector and pc_dealloc will be posted with a pixel vector unless none is in progress. In this case the pc_dealloc will be posted in the request queue. Filler quads will be transferred with The Quad mask set but the pixel corresponding pixel mask set to zero.
SC_SQ_valid	1	SC sending valid data, 2nd clk could be all zeroes

SC_SQ_data – first clock and second clock transfers are shown in the table below.

Name	**BitField**	**Bits**	**Description**
1st Clock Transfer			
SC_SQ_event	0	1	This transfer is a 1 clock event vector Force quad_mask = new_vector=pc_dealloc=0
SC_SQ_event_id	[5:1]	4	This field identifies the event 0 => denotes an End Of State Event 1

			=> TBD
SC_SQ_state_id	[8:6]	3	State/constant pointer (6*3+3)
SC_SQ_pc_dealloc	[11:9]	3	Deallocation token for the Parameter Cache
SC_SQ_new_vector	12	1	The SQ must wait for Vertex shader done count > 0 and after dispatching the Pixel Vector the SQ will decrement the count.
SC_SQ_quad_mask	[16:13]	4	Quad Write mask left to right SP0 => SP3
SC_SQ_end_of_prim	17	1	End Of the primitive
SC_SQ_pix_mask	[33:18]	16	Valid bits for all pixels SP0=>SP3 (UL,UR,LL,LR)
SC_SQ_provok_vtx	[35:34]	2	Provoking vertex for flat shading
SC_SQ_lod_correct_0	[44:36]	9	LOD correction for quad 0 (SP0) (9 bits per quad)
SC_SQ_lod_correct_1	[53:45]	9	LOD correction for quad 1 (SP1) (9 bits per quad)
2nd Clock Transfer			
SC_SQ_lod_correct_2	[8:0]	9	LOD correction for quad 2 (SP2) (9 bits per quad)
SC_SQ_lod_correct_3	[17:9]	9	LOD correction for quad 3 (SP3) (9 bits per quad)
SC_SQ_pc_ptr0	[28:18]	11	Parameter Cache pointer for vertex 0
SC_SQ_pc_ptr1	[39:29]	11	Parameter Cache pointer for vertex 1
SC_SQ_pc_ptr2	[50:40]	11	Parameter Cache pointer for vertex 2
SC_SQ_prim_type	[53:51]	3	Stippled line and Real time command need to load tex cords from alternate buffer 000: Sprite (point) 001: Line 010: Tri_rect 100: Realtime Sprite (point) 101: Realtime Line 110: Realtime Tri_rect

Name	**Bits**	**Description**
SQ_SC_free_buff	1	Pipelined bit that instructs SC to decrement count of buffers in use.
SQ_SC_dec_cntr_cnt	1	Pipelined bit that instructs SC to decrement count of new vector and/or event sent to prevent SC from overflowing SQ interpolator/Reservation request fifo.

The scan converter will submit a partial vector whenever:

1.) He gets a primitive marked with an end of packet signal.

2.) A current pixel vector is being assembled with at least one or more valid quads and the vector has been marked for deallocate when a primitive marked new_vector arrives. The Scan Converter will submit a partial vector (up to 16quads with zero pixel mask to fill out the vector) prior to submitting the new_vector marker\primitive.

(This will prevent a hang which can be demonstrated when all primitives in a packet three vectors are culled except for a one quad primitive that gets marked pc_dealloc (vertices maximum size). In this case two new_vectors are submitted and processed, but then one valid quad with the pc_dealloc creates a vector and then the new would wait for another vertex vector to be processed, but the one being waited for could never export until the pc_dealloc signal made it through and thus the hang.)

23.2.3 SQ to SX(SP): Interpolator bus

Name	Direction	Bits	Description
SQ_SPx_interp_flat_vtx	SQ→SPx	2	Provoking vertex for flat shading
SQ_SPx_interp_flat_gouraud	SQ→SPx	1	Flat or gouraud shading
SQ_SPx_interp_cyl_wrap	SQ→SPx	4	Wich channel needs to be cylindrical wrapped
SQ_SPx_interp_param_gen	SQ→SPx	1	Generate Parameter
SQ_SPx_interp_prim_type	SQ→SPx	2	Bits [1:0] of primitive type sent by SC
SQ_SPx_interp_buff_swap	SQ→SPx	1	Swapp IJ buffers
SQ_SPx_interp_IJ_line	SQ→SPx	2	IJ line number
SQ_SPx_interp_mode	SQ→SPx	1	Center/Centroid sampling
SQ_SXx_pc_ptr0	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_pc_ptr1	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_pc_ptr2	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_rt_sel	SQ→SXx	1	Selects between RT and Normal data (Bit 2 of prim type)
SQ_SX0_pc_wr_en	SQ→SX0	8	Write enable for the PC memories
SQ_SX1_pc_wr_en	SQ→SX1	8	Write enable for the PC memories
SQ_SXx_pc_wr_addr	SQ→SXx	7	Write address for the PCs
SQ_SXx_pc_channel_mask	SQ→SXx	4	Channel mask
SQ_SXx_pc_ptr_valid	SQ→SXx	1	Read pointers are valid.
SQ_SPx_interp_valid	SQ→SPx	1	Interpolation control valid

23.2.4 SQ to SP: Staging Register Data

This is a broadcast bus that sends the VSISR information to the staging registers of the shader pipes.

Name	Direction	Bits	Description
SQ_SPx_vsr_data	SQ→SPx	96	Pointers of indexes or HOS surface information
SQ_SPx_vsr_double	SQ→SPx	1	0: Normal 96 bits per vert 1: double 192 bits per vert
SQ_SP0_ vsr_valid	SQ→SP0	1	Data is valid
SQ_SP1_ vsr_ valid	SQ→SP1	1	Data is valid
SQ_SP2_ vsr_ valid	SQ→SP2	1	Data is valid
SQ_SP3_ vsr_ valid	SQ→SP3	1	Data is valid
SQ_SPx_vsr_read	SQ→SPx	1	Increment the read pointers

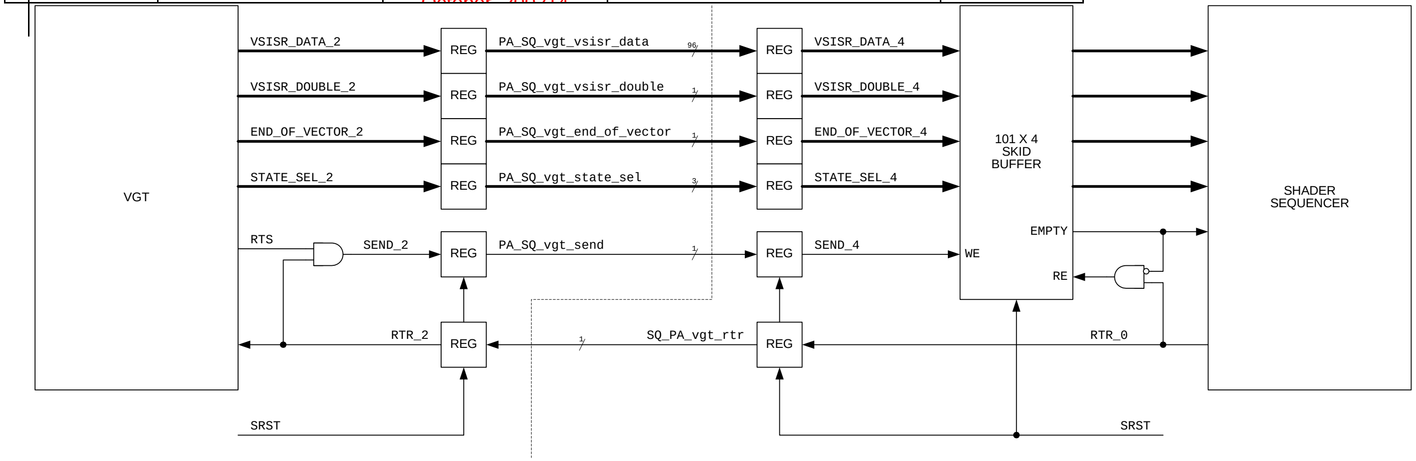
23.2.5 VGT to SQ : Vertex interface

23.2.5.1 Interface Signal Table

The area difference between the two methods is not sufficient to warrant complicating the interface or the state requirements of the VSISRs. **Therefore, the POR for this interface is that the VGT will transmit the data to the VSISRs (via the Shader Sequencer) in full, 32-bit floating-point format.** The VGT can transmit up to six 32-bit floating-point values to each VSISR where four or more values require two transmission clocks. The data bus is 96 bits wide. In the case where an event is sent the 5 LSBs of VGT_SQ_vsisr_data contain the eventID.

Name	Bits	Description
VGT_SQ_vsisr_data	96	Pointers of indexes or HOS surface information
VGT_SQ_event	1	VGT is sending an event
VGT_SQ_vsisr_continued	1	0: Normal 96 bits per vert 1: double 192 bits per vert
VGT_SQ_end_of_vtx_vect	1	Indicates the last VSISR data set for the current process vector (for double vector data, "end_of_vector" is set on the first vector)
VGT_SQ_indx_valid	1	Vsisr data is valid
VGT_SQ_state	3	Render State (6*3+3 for constants). This signal is guaranteed to be correct when "VGT_SQ_vgt_end_of_vector" is high.
VGT_SQ_send	1	Data on the VGT_SQ is valid receive (see write-up for standard R400 SEND/RTR interface handshaking)
SQ_VGT_rtr	1	Ready to receive (see write-up for standard R400 SEND/RTR interface handshaking)

23.2.5.2 Interface Diagrams



VGT
VSISR_DATA_2
VSISR_DOUBLE_2
END_OF_VECTOR_2
STATE_SEL_2
RTS
SEND_2
RTR_2
SRST
REG
PA_SQ_vgt_vsisr_data
96
PA_SQ_vgt_vsisr_double
1
PA_SQ_vgt_end_of_vector
1
PA_SQ_vgt_state_sel
3
PA_SQ_vgt_send
1
SQ_PA_vgt_rtr
1
VSISR_DATA_4
VSISR_DOUBLE_4
END_OF_VECTOR_4
STATE_SEL_4
SEND_4
101 X 4
SKID
BUFFER
EMPTY
WE
RE
RTR_0
SRST
SHADER
SEQUENCER

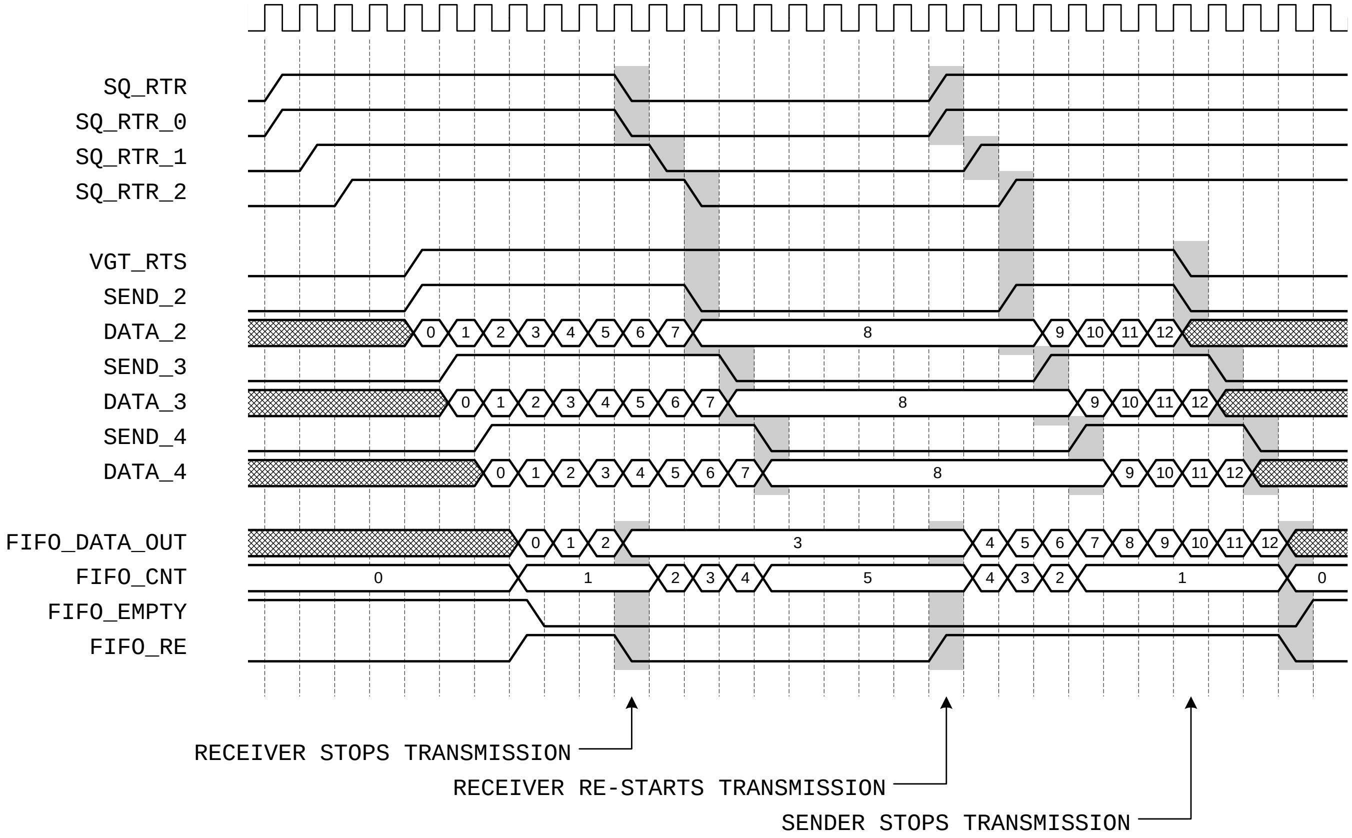


Figure 1. Detailed Logical Diagram for PA_SQ_vgt Interface.

23.2.6 *SQ to SX: Control bus*

Name	Direction	Bits	Description
SQ_SXx_exp_type	SQ→SXx	2	00: Pixel without z (1 to 4 buffers) 01: Pixel with z (1 to 4 buffers) 10: Position (1 or 2 results) 11: Pass thru (4,8 or 12 results aligned)
SQ_SXx_exp_number	SQ→SXx	2	Number of locations needed in the export buffer (encoding depends on the type see bellow).
SQ_SXx_exp_alu_id	SQ→SXx	1	ALU ID
SQ_SXx_exp_valid	SQ→SXx	1	Valid bit
SQ_SXx_exp_state	SQ→SXx	3	State Context
SQ_SXx_free_done	SQ→SXx	1	Pulse that indicates that the previous export is finished **from the point of view of the SP. This does not necessarily mean that the data has been transferred to RB or PA, or that the space in export buffer for that particular vector thread has been freed up.**
SQ_SXx_free_alu_id	SQ→SXx	1	ALU ID

Depending on the type the number of export location changes:

- Type 00 : Pixels without Z
  - 00 = 1 buffer
  - 01 = 2 buffers
  - 10 = 3 buffers
  - 11 = 4 buffer
- Type 01: Pixels with Z
  - 00 = 2 Buffers (color + Z)
  - 01 = 3 buffers (2 color + Z)
  - 10 = 4 buffers (3 color + Z)
  - 11 = 5 buffers (4 color + Z)
- Type 10 : Position export
  - 00 = 1 position
  - 01 = 2 positions
  - 1X = Undefined
- Type 11: Pass Thru
  - 00 = 4 buffers
  - 01 = 8 buffers
  - 10 = 12 buffers
  - 11 = Undefined

Below the thick black line is the end of transfer packet that tells the SX that a given export is finished. The report packet **will always arrive either before or at the same time than the next export to the same ALU id**.

23.2.7 *SX to SQ : Output file control*

Name	Direction	Bits	Description
SXx_SQ_exp_count_rdy	SXx→SQ	1	Raised by SX0 to indicate that the following two fields reflect the result of the most recent export
SXx_SQ_exp_pos_avail	SXx→SQ	2	Specifies whether there is room for another position. 00 : 0 buffers ready 01 : 1 buffer ready 10 : 2 or more buffers ready
SXx_SQ_exp_buf_avail	SXx→SQ	7	Specifies the space available in the output buffers. 0: buffers are full 1: 2K-bits available (32-bits for each of the 64

			pixels in a clause) ... 64: 128K-bits available (16 128-bit entries for each of 64 pixels) 65-127: RESERVED

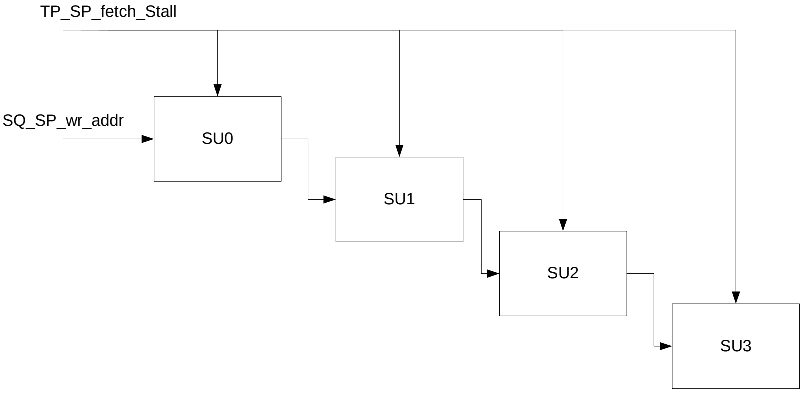
23.2.8 *SQ to TP: Control bus*

Once every clock, the fetch unit sends to the sequencer on which RS line it is now working and if the data in the GPRs is ready or not. This way the sequencer can update the fetch valid bits flags for the reservation station. The sequencer also provides the instruction and constants for the fetch to execute and the address in the register file where to write the fetch return data.

Name	Direction	Bits	Description
TPx_SQ_data_rdy	TPx→ SQ	1	Data ready
TPx_SQ_rs_line_num	TPx→ SQ	6	Line number in the Reservation station
TPx_SQ_type	TPx→ SQ	1	Type of data sent (0:PIXEL, 1:VERTEX)
SQ_TPx_send	SQ→TPx	1	Sending valid data
SQ_TPx_const	SQ→TPx	48	Fetch state sent over 4 clocks (192 bits total)
SQ_TPx_instr	SQ→TPx	24	Fetch instruction sent over 4 clocks
SQ_TPx_end_of_group	SQ→TPx	1	Last instruction of the group
SQ_TPx_Type	SQ→TPx	1	Type of data sent (0:PIXEL, 1:VERTEX)
SQ_TPx_gpr_phase	SQ→TPx	2	Write phase signal
SQ_TP0_lod_correct	SQ→TP0	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP0_pix_mask	SQ→TP0	4	Pixel mask 1 bit per pixel
SQ_TP1_lod_correct	SQ→TP1	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP1_pix_mask	SQ→TP1	4	Pixel mask 1 bit per pixel
SQ_TP2_lod_correct	SQ→TP2	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP2_pix_mask	SQ→TP2	4	Pixel mask 1 bit per pixel
SQ_TP3_lod_correct	SQ→TP3	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP3_pix_mask	SQ→TP3	4	Pixel mask 1 bit per pixel
SQ_TPx_rs_line_num	SQ→TPx	6	Line number in the Reservation station
SQ_TPx_write_gpr_index	SQ->TPx	7	Index into Register file for write of returned Fetch Data
SQ_TPx_ctx_id	SQ→TPx	3	The state context ID (needed for multisample resolves)

23.2.9 *TP to SQ: Texture stall*

The TP sends this signal to the SQ and the SPs when its input buffer is full.



Name	Direction	Bits	Description
TP_SQ_fetch_stall	TP→ SQ	1	Do not send more texture request if asserted

23.2.10 SQ to SP: Texture stall

Name	Direction	Bits	Description
SQ_SPx_fetch_stall	SQ→SPx	1	Do not send more texture request if asserted

23.2.11 SQ to SP: GPR and auto counter

Name	Direction	Bits	Description
SQ_SPx_gpr_wr_addr	SQ→SPx	7	Write address
SQ_SPx_gpr_rd_addr	SQ→SPx	7	Read address
SQ_SPx_gpr_rd_en	SQ→SPx	1	Read Enable
SQ_SP0_gpr_wr_en	SQ→SPx	4	Write Enable for the GPRs of SP0
SQ_SP1_gpr_wr_en	SQ→SPx	4	Write Enable for the GPRs of SP1
SQ_SP2_gpr_wr_en	SQ→SPx	4	Write Enable for the GPRs of SP2
SQ_SP3_gpr_wr_en	SQ→SPx	4	Write Enable for the GPRs of SP3
SQ_SPx_gpr_phase	SQ→SPx	2	The phase mux (arbitrates between inputs, ALU SRC reads and writes)
SQ_SPx_channel_mask	SQ→SPx	4	The channel mask
SQ_SPx_gpr_input_sel	SQ→SPx	2	When the phase mux selects the inputs this tells from which source to read from: Interpolated data, VTX0, VTX1, autogen counter.
SQ_SPx_auto_count	SQ→SPx	21	Auto count generated by the SQ, common for all shader pipes

23.2.12 *SQ to SPx: Instructions*

Name	Direction	Bits	Description
SQ_SPx_instr_start	SQ→SPx	1	Instruction start
SQ_SP_instr	SQ→SPx	24	Transferred over 4 cycles 0: SRC A Negate Argument Modifier 0:0 SRC A Abs Argument Modifier 1:1 SRC A Swizzle 9:2 Vector Dst 15:10 Per channel Select 23:16 00: GPR 01: PV 10: PS 11: Constant (if 11 has to be 11 for all channels) -- - 1: SRC B Negate Argument Modifier 0:0 SRC B Abs Argument Modifier 1:1 SRC B Swizzle 9:2 Scalar Dst 15:10 Per channel Select 23:16 00: GPR 01: PV 10: PS 11: Constant (if 11 has to be 11 for all channels) -- - 2: SRC C Negate Argument Modifier 0:0 SRC C Abs Argument Modifier 1:1 SRC C Swizzle 9:2 Unused 15:10 Per channel Select 23:16 00: GPR 01: PV 10: PS 11: Constant (if 11 has to be 11 for all channels) -- - 3: Vector Opcode 4:0 Scalar Opcode 10:5 Vector Clamp 11:11 Scalar Clamp 12:12 Vector Write Mask 16:13 Scalar Write Mask 20:17 Unused 23:21
SQ_SP0_pred_override	SQ→SP0	4	0: Use per channel RGBA field (enables the per channel logic, if not set only pay attention to the 11 seting). 1: Use GPR
SQ_SP1_pred_override	SQ→SP1	4	0: Use per channel RGBA field (enables the per channel logic, if not set only pay attention to the 11 seting). 1: Use GPR
SQ_SP2_pred_override	SQ→SP2	4	0: Use per channel RGBA field (enables the per channel logic, if not set only pay attention to the 11 seting). 1: Use GPR
SQ_SP3_pred_override	SQ→SP3	4	0: Use per channel RGBA field (enables the per channel logic, if not set only pay attention to the 11 seting). 1: Use GPR
SQ_SPx_exp_id	SQ→SPx	1	GPR ID

SQ_SPx_exporting	SQ→SPx	1	0: Not Exporting 1: Exporting
SQ_SPx_stall	SQ→SPx	1	Stall signal

23.2.13 SQ to SX: write mask interface (must be aligned with the SP data)

Name	Direction	Bits	Description
SQ_SX0_write_mask	SQ→SP0	8	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock. This is for the data coming of SP0 and SP2.
SQ_SX1_ write_mask	SQ→SP1	8	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock. This is for the data coming of SP1 and SP3.

23.2.14 SP to SQ: Constant address load/ Predicate Set/Kill set

Name	Direction	Bits	Description
SP0_SQ_const_addr	SP0→SQ	36	Constant address load / predicate vector load (4 bits only)/ Kill vector load (4 bits only) to the sequencer
SP0_SQ_valid	SP0→SQ	1	Data valid
SP1_SQ_const_addr	SP1→SQ	36	Constant address load / predicate vector load (4 bits only)/ Kill vector load (4 bits only) to the sequencer
SP1_SQ_valid	SP1→SQ	1	Data valid
SP2_SQ_const_addr	SP2→SQ	36	Constant address load / predicate vector load (4 bits only)/ Kill vector load (4 bits only) to the sequencer
SP2_SQ_valid	SP2→SQ	1	Data valid
SP3_SQ_const_addr	SP3→SQ	36	Constant address load / predicate vector load (4 bits only)/ Kill vector load (4 bits only) to the sequencer
SP3_SQ_valid	SP3→SQ	1	Data valid
SP0_SQ_data_type	SP→SQ	2	Data Type 0: Constant Load 1: Predicate Set 2: Kill vector load

Because of the sharing of the bus none of the MOVA, PREDSET or KILL instructions may be coissued.

23.2.15 SQ to SPx: constant broadcast

Name	Direction	Bits	Description
SQ_SPx_const	SQ→SPx	128	Constant broadcast

23.2.16 SQ to CP: RBBM bus

Name	Direction	Bits	Description
SQ_RBB_rs	SQ→CP	1	Read Strobe
SQ_RBB_rd	SQ→CP	32	Read Data
SQ_RBBM_nrtrtr	SQ→CP	1	Optional
SQ_RBBM_rtr	SQ→CP	1	Real-Time (Optional)

23.2.17 CP to SQ: RBBM bus

Name	Direction	Bits	Description
rbbm_we	CP→SQ	1	Write Enable
rbbm_a	CP→SQ	15	Address -- Upper Extent is TBD (16:2)
rbbm_wd	CP→SQ	32	Data
rbbm_be	CP→SQ	4	Byte Enables
rbbm_re	CP→SQ	1	Read Enable
rbb_rs0	CP→SQ	1	Read Return Strobe 0

rbb_rs1	CP→SQ	1	Read Return Strobe 1
rbb_rd0	CP→SQ	32	Read Data 0
rbb_rd1	CP→SQ	32	Read Data 0
RBBM_SQ_soft_reset	CP→SQ	1	Soft Reset

23.2.18 *SQ to CP: State report*

Name	Direction	Bits	Description
SQ_CP_vs_event	SQ→CP	1	Vertex Shader Event
SQ_CP_vs_eventid	SQ→CP	5	Vertex Shader Event ID
SQ_CP_ps_event	SQ→CP	1	Pixel Shader Event
SQ_CP_ps_eventid	SQ→CP	5	Pixel Shader Event ID

23.3 Example of control flow program execution

We now provide some examples of execution to better illustrate the new design.

Given the program:

Alu 0
Alu 1
Tex 0
Tex 1
Alu 3 Serial
Alu 4
Tex 2
Alu 5
Alu 6 Serial
Tex 3
Alu 7
Alloc Position 1 buffer
Alu 8 Export
Tex 4
Alloc Parameter 3 buffers
Alu 9 Export 0
Tex 5
Alu 10 Serial Export 2
Alu 11 Export 1 End

Would be converted into the following CF instructions:

```
Execute 0 Alu 0 Alu 0 Tex 0 Tex 1 Alu 0 Alu 0 Tex 0 Alu 1 Alu 0 Tex
Execute 0 Alu
Alloc Position 1
Execute 0 Alu 0 Tex
Alloc Param 3
Execute_end 0 Alu 0 Tex 1 Alu 0 Alu
```

And the execution of this program would look like this:

Put thread in Vertex RS:

Control Flow Instruction Pointer (12 bits), (CFP)
Execution Count Marker (3 or 4 bits), (ECM)
Loop Iterators (4x9 bits), (LI)
Call return pointers (4x12 bits), (CRP)
Predicate Bits(4x64 bits), (PB)
Export ID (1 bit), (EXID)
GPR Base Ptr (8 bits), (GPR)

Export Base Ptr (7 bits), (EB)
Context Ptr (3 bits).(CPTR)
LOD correction bits (16x6 bits) (LOD)

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	0	0	0	0	0	0	0	0	0

Valid Thread (VALID)
Texture/ALU engine needed (TYPE)
Texture Reads are outstanding (PENDING)
Waiting on Texture Read to Complete (SERIAL)
Allocation Wait (2 bits) (ALLOC)
  00 – No allocation needed
  01 – Position export allocation needed (ordered export)
  10 – Parameter or pixel export needed (ordered export)
  11 – pass thru (out of order export)
Allocation Size (4 bits) (SIZE)
Position Allocated (POS_ALLOC)
First thread of a new context (FIRST)
Last (1 bit), (LAST)

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	0	0	0	0	0	1	0

Then the thread is picked up for the execution of the first control flow instruction:

```
Execute 0 Alu 0 Alu 0 Tex 0 Tex 1 Alu 0 Alu 0 Tex 0 Alu 1 Alu 0 Tex
```

It executes the first two ALU instructions and goes back to the RS for a resource request change. Here is the state returned to the RS:

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	2	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	0	0	0	0	0	1	0

Then when the texture pipe frees up, the arbiter picks up the thread to issue the texture reads. The thread comes back in this state:

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	4	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	1	0	0	0	1	0

Because of the serial bit the arbiter must wait for the texture to return and clear the PENDING bit before it can pick the thread up. Lets say that the texture reads are complete, then the arbiter picks up the thread and returns it in this state:

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	6	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	0	0	0	0	0	1	0

Again the TP frees up, the arbiter picks up the thread and executes. It returns in this state:

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	7	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	0	0	0	1	0

Now, even if the texture has not returned we can still pick up the thread for ALU execution because the serial bit is not set. The thread will however come back to the RS for the second ALU instruction because it has the serial bit set.

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	8	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	1	0	0	0	1	0

As soon as the TP clears the pending bit the thread is picked up and returns:

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	9	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	0	0	0	0	0	1	0

Picked up by the TP and returns:

```
Execute 0 Alu
```

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
1	0	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	0	0	0	1	0

Picked up by the ALU and returns (lets say the TP has not returned yet):

```
Alloc Position 1
```

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
2	0	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	01	1	0	1	0

If the SX has the place for the export, the SQ is going to allocate and pick up the thread for execution. It returns to the RS in this state:

```
Execute 0 Alu 0 Tex
```

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
3	1	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	1	0	0	0	1	1	0

Now, since the TP has not returned yet, we must wait for it to return because we cannot issue multiple texture requests. The TP returns, clears the PENDING bit and we proceed:

```
Alloc Param 3
```

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
4	0	0	0	0	1	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	10	3	1	1	0

Once again the SQ makes sure the SX has enough room in the Parameter cache before it can pick up this thread.

```
Execute_end 0 Alu 0 Tex 1 Alu 0 Alu
```

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
5	1	0	0	0	1	0	100	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	1	0	0	0	1	1	0

This executes on the TP and then returns:

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
5	2	0	0	0	1	0	100	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	1	0	0	1	1	1

Waits for the TP to return because of the textures reads are pending (and SERIAL in this case). Then executes and does not return to the RS because the LAST bit is set. This is the end of this thread and before dropping it on the floor, the SQ notifies the SX of export completion.

24. Open issues

Need to do some testing on the size of the register file as well as on the register file allocation method (dynamic VS static).

Saving power?

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 2015~~7~~ January, 200316	GEN-CXXXXX-REVA	1 of 54

Author: Laurent Lefebvre

Issue To:	**Copy No:**

R400 Sequencer Specification

SQ

Version 2.09~~8~~

Overview: This is an architectural specification for the R400 Sequencer block (SEQ). It provides an overview of the required capabilities and expected uses of the block. It also describes the block interfaces, internal sub-blocks, and provides internal state diagrams.

AUTOMATICALLY UPDATED FIELDS:

Document Location:	C:\perforce\r400\doc_lib\design\blocks\sq\R400_Sequencer.doc
Current Intranet Search Title:	R400 Sequencer Specification

APPROVALS

Name/Dept	Signature/Date

Remarks:

THIS DOCUMENT CONTAINS CONFIDENTIAL INFORMATION THAT COULD BE SUBSTANTIALLY DETRIMENTAL TO THE INTEREST OF ATI TECHNOLOGIES INC. THROUGH UNAUTHORIZED USE OR DISCLOSURE.

Table Of Contents

1. OVERVIEW .... 8~~7~~
1.1 Top Level Block Diagram .... 10~~9~~
1.2 Data Flow graph (SP) .... 11~~10~~
1.3 Control Graph .... 12~~11~~
2. INTERPOLATED DATA BUS .... 12~~11~~
3. INSTRUCTION STORE .... 15~~14~~
4. SEQUENCER INSTRUCTIONS .... 15~~14~~
5. CONSTANT STORES .... 15~~14~~
5.1 Memory organizations .... 15~~14~~
5.2 Management of the Control Flow Constants .... 16~~15~~
5.3 Management of the re-mapping tables .... 16~~15~~
5.3.1 R400 Constant management .... 16~~15~~
5.3.2 Proposal for R400LE constant management .... 16~~15~~
5.3.3 Dirty bits .... 18~~17~~
5.3.4 Free List Block .... 18~~17~~
5.3.5 De-allocate Block .... 19~~18~~
5.3.6 Operation of Incremental model .... 19~~18~~
5.4 Constant Store Indexing .... 19~~18~~
5.5 Real Time Commands .... 20~~19~~
5.6 Constant Waterfalling .... 20~~19~~
6. LOOPING AND BRANCHES .... 21~~20~~
6.1 The controlling state. .... 21~~20~~
6.2 The Control Flow Program .... 21~~20~~
6.2.1 Control flow instructions table .... 22~~21~~
6.3 Implementation .... 24~~23~~
6.4 Data dependant predicate instructions .... 26~~24~~
6.5 HW Detection of PV,PS .... 26~~25~~
6.6 Register file indexing .... 26~~25~~
6.7 Debugging the Shaders .... 27~~26~~
6.7.1 Method 1: Debugging registers .... 27~~26~~
6.7.2 Method 2: Exporting the values in the GPRs .... 27~~26~~
7. PIXEL KILL MASK .... 27~~26~~
8. MULTIPASS VERTEX SHADERS (HOS) .... 27~~26~~
9. REGISTER FILE ALLOCATION .... 28~~26~~
10. FETCH ARBITRATION .... 29~~28~~
11. ALU ARBITRATION .... 29~~28~~
12. HANDLING STALLS .... 30~~29~~
13. CONTENT OF THE RESERVATION STATION FIFOS .... 30~~29~~
14. THE OUTPUT FILE .... 30~~29~~
15. IJ FORMAT .... 30~~29~~
15.1 Interpolation of constant attributes .... 30~~29~~
16. STAGING REGISTERS .... 31~~30~~

17. THE PARAMETER CACHE .... 32~~31~~
17.1 Export restrictions .... 33~~32~~
17.1.1 Pixel exports: .... 33~~32~~
17.1.2 Vertex exports: .... 33~~32~~
17.1.3 Pass thru exports: .... 33~~32~~
17.2 Arbitration restrictions .... 34~~32~~
18. EXPORT TYPES .... 34~~32~~
18.1 Vertex Shading .... 34~~32~~
18.2 Pixel Shading .... 34~~33~~
19. SPECIAL INTERPOLATION MODES .... 35~~33~~
19.1 Real time commands .... 35~~33~~
19.2 Sprites/ XY screen coordinates/ FB information .... 35~~33~~
19.3 Auto generated counters .... 36~~34~~
19.3.1 Vertex shaders .... 36~~34~~
19.3.2 Pixel shaders .... 36~~34~~
20. STATE MANAGEMENT .... 37~~35~~
20.1 Parameter cache synchronization .... 37~~35~~
21. XY ADDRESS IMPORTS .... 37~~35~~
21.1 Vertex indexes imports .... 37~~35~~
22. REGISTERS .... 37~~35~~
23. INTERFACES .... 38~~36~~
23.1 External Interfaces .... 38~~36~~
23.2 SC to SP Interfaces .... 38~~36~~
23.2.1 SC_SP# .... 38~~36~~
23.2.2 SC_SQ .... 39~~37~~
23.2.3 SQ to SX(SP): Interpolator bus .... 41~~39~~
23.2.4 SQ to SP: Staging Register Data .... 41~~39~~
23.2.5 VGT to SQ : Vertex interface .... 41~~39~~
23.2.6 SQ to SX: Control bus .... 45~~42~~
23.2.7 SX to SQ : Output file control .... 45~~42~~
23.2.8 SQ to TP: Control bus .... 46~~43~~
23.2.9 TP to SQ: Texture stall .... 46~~43~~
23.2.10 SQ to SP: Texture stall .... 47~~44~~
23.2.11 SQ to SP: GPR and auto counter .... 47~~44~~
23.2.12 SQ to SPx: Instructions .... 48~~45~~
23.2.13 SQ to SX: write mask interface (must be aligned with the SP data) .... 49~~46~~
23.2.14 SP to SQ: Constant address load/ Predicate Set/Kill set .... 49~~46~~
23.2.15 SQ to SPx: constant broadcast .... 49~~46~~
23.2.16 SQ to CP: RBBM bus .... 49~~46~~
23.2.17 CP to SQ: RBBM bus .... 50~~46~~
23.2.18 SQ to CP: State report .... 50~~47~~

23.3 Example of control flow program execution .... 50~~47~~
24. OPEN ISSUES .... 54~~51~~
~~1. OVERVIEW .... 7~~
~~1.1 Top Level Block Diagram .... 9~~
~~1.2 Data Flow graph (SP) .... 10~~
~~1.3 Control Graph .... 11~~
~~2. INTERPOLATED DATA BUS .... 11~~
~~3. INSTRUCTION STORE .... 14~~
~~4. SEQUENCER INSTRUCTIONS .... 14~~
~~5. CONSTANT STORES .... 14~~
~~5.1 Memory organizations .... 14~~
~~5.2 Management of the Control Flow Constants .... 15~~
~~5.3 Management of the re-mapping tables .... 15~~
~~5.3.1 R400 Constant management .... 15~~
~~5.3.2 Proposal for R400LE constant management .... 15~~
~~5.3.3 Dirty bits .... 17~~
~~5.3.4 Free List Block .... 17~~
~~5.3.5 De-allocate Block .... 18~~
~~5.3.6 Operation of Incremental model .... 18~~
~~5.4 Constant Store Indexing .... 18~~
~~5.5 Real Time Commands .... 19~~
~~5.6 Constant Waterfalling .... 19~~
~~6. LOOPING AND BRANCHES .... 20~~
~~6.1 The controlling state. .... 20~~
~~6.2 The Control Flow Program .... 20~~
~~6.2.1 Control flow instructions table .... 21~~
~~6.3 Implementation .... 23~~
~~6.4 Data dependant predicate instructions .... 24~~
~~6.5 HW Detection of PV,PS .... 25~~
~~6.6 Register file indexing .... 25~~
~~6.7 Debugging the Shaders .... 25~~
~~6.7.1 Method 1: Debugging registers .... 26~~
~~6.7.2 Method 2: Exporting the values in the GPRs .... 26~~
~~7. PIXEL KILL MASK .... 26~~
~~8. MULTIPASS VERTEX SHADERS (HOS) .... 26~~
~~9. REGISTER FILE ALLOCATION .... 26~~
~~10. FETCH ARBITRATION .... 28~~
~~11. ALU ARBITRATION .... 28~~
~~12. HANDLING STALLS .... 29~~
~~13. CONTENT OF THE RESERVATION STATION FIFOS .... 29~~
~~14. THE OUTPUT FILE .... 29~~
~~15. IJ FORMAT .... 29~~
~~15.1 Interpolation of constant attributes .... 29~~
~~16. STAGING REGISTERS .... 30~~
~~17. THE PARAMETER CACHE .... 31~~
~~17.1 Export restrictions .... 32~~

~~17.1.1 Pixel exports: 32~~
~~17.1.2 Vertex exports: 32~~
~~17.1.3 Pass thru exports: 32~~
~~17.2 Arbitration restrictions 32~~
~~**18. EXPORT TYPES 32**~~
~~18.1 Vertex Shading 32~~
~~18.2 Pixel Shading 33~~
~~**19. SPECIAL INTERPOLATION MODES 33**~~
~~19.1 Real time commands 33~~
~~19.2 Sprites/ XY screen coordinates/ FB information 33~~
~~19.3 Auto generated counters 34~~
~~19.3.1 Vertex shaders 34~~
~~19.3.2 Pixel shaders 34~~
~~**20. STATE MANAGEMENT 35**~~
~~20.1 Parameter cache synchronization 35~~
~~**21. XY ADDRESS IMPORTS 35**~~
~~21.1 Vertex indexes imports 35~~
~~**22. REGISTERS 35**~~
~~**23. INTERFACES 36**~~
~~23.1 External Interfaces 36~~
~~23.2 SC to SP Interfaces 36~~
~~23.2.1 SC_SP# 36~~
~~23.2.2 SC_SQ 37~~
~~23.2.3 SQ to SX(SP): Interpolator bus 39~~
~~23.2.4 SQ to SP: Staging Register Data 39~~
~~23.2.5 VGT to SQ : Vertex interface 39~~
~~23.2.6 SQ to SX: Control bus 42~~
~~23.2.7 SX to SQ : Output file control 42~~
~~23.2.8 SQ to TP: Control bus 43~~
~~23.2.9 TP to SQ: Texture stall 43~~
~~23.2.10 SQ to SP: Texture stall 44~~
~~23.2.11 SQ to SP: GPR and auto counter 44~~
~~23.2.12 SQ to SPx: Instructions 45~~
~~23.2.13 SP to SQ: Constant address load/ Predicate Set/Kill set 46~~
~~23.2.14 SQ to SPx: constant broadcast 46~~
~~23.2.15 SQ to CP: RBBM bus 46~~
~~23.2.16 CP to SQ: RBBM bus 46~~
~~23.2.17 SQ to CP: State report 47~~
~~23.3 Example of control flow program execution 47~~
~~**24. OPEN ISSUES 51**~~

Revision Changes:

Rev 0.1 (Laurent Lefebvre) Date: May 7, 2001	First draft.
Rev 0.2 (Laurent Lefebvre) Date : July 9, 2001	Changed the interfaces to reflect the changes in the SP. Added some details in the arbitration section.
Rev 0.3 (Laurent Lefebvre) Date : August 6, 2001	Reviewed the Sequencer spec after the meeting on August 3, 2001.
Rev 0.4 (Laurent Lefebvre) Date : August 24, 2001	Added the dynamic allocation method for register file and an example (written in part by Vic) of the flow of pixels/vertices in the sequencer.
Rev 0.5 (Laurent Lefebvre) Date : September 7, 2001	Added timing diagrams (Vic)
Rev 0.6 (Laurent Lefebvre) Date : September 24, 2001	Changed the spec to reflect the new R400 architecture. Added interfaces.
Rev 0.7 (Laurent Lefebvre) Date : October 5, 2001	Added constant store management, instruction store management, control flow management and data dependant predication.
Rev 0.8 (Laurent Lefebvre) Date : October 8, 2001	Changed the control flow method to be more flexible. Also updated the external interfaces.
Rev 0.9 (Laurent Lefebvre) Date : October 17, 2001	Incorporated changes made in the 10/18/01 control flow meeting. Added a NOP instruction, removed the conditional_execute_or_jump. Added debug registers.
Rev 1.0 (Laurent Lefebvre) Date : October 19, 2001	Refined interfaces to RB. Added state registers.
Rev 1.1 (Laurent Lefebvre) Date : October 26, 2001	Added SEQ→SP0 interfaces. Changed delta precision. Changed VGT→SP0 interface. Debug Methods added.
Rev 1.2 (Laurent Lefebvre) Date : November 16, 2001	Interfaces greatly refined. Cleaned up the spec.
Rev 1.3 (Laurent Lefebvre) Date : November 26, 2001	Added the different interpolation modes.
Rev 1.4 (Laurent Lefebvre) Date : December 6, 2001	Added the auto incrementing counters. Changed the VGT→SQ interface. Added content on constant management. Updated GPRs.
Rev 1.5 (Laurent Lefebvre) Date : December 11, 2001	Removed from the spec all interfaces that weren't directly tied to the SQ. Added explanations on constant management. Added PA→SQ synchronization fields and explanation.
Rev 1.6 (Laurent Lefebvre) Date : January 7, 2002	Added more details on the staging register. Added detail about the parameter caches. Changed the call instruction to a Conditionnal_call instruction. Added details on constant management and updated the diagram.
Rev 1.7 (Laurent Lefebvre) Date : February 4, 2002	Added Real Time parameter control in the SX interface. Updated the control flow section.
Rev 1.8 (Laurent Lefebvre) Date : March 4, 2002	New interfaces to the SX block. Added the end of clause modifier, removed the end of clause instructions.
Rev 1.9 (Laurent Lefebvre) Date : March 18, 2002	Rearangement of the CF instruction bits in order to ensure byte alignement.
Rev 1.10 (Laurent Lefebvre) Date : March 25, 2002	Updated the interfaces and added a section on exporting rules.
Rev 1.11 (Laurent Lefebvre) Date : April 19, 2002	Added CP state report interface. Last version of the spec with the old control flow scheme
Rev 2.0 (Laurent Lefebvre) Date : April 19, 2002	New control flow scheme

Rev 2.01 (Laurent Lefebvre) Date : May 2, 2002	Changed slightly the control flow instructions to allow force jumps and calls.
Rev 2.02 (Laurent Lefebvre) Date : May 13, 2002	Updated the Opcodes. Added type field to the constant/pred interface. Added Last field to the SQ→SP instruction load interface.
Rev 2.03 (Laurent Lefebvre) Date : July 15, 2002	SP interface updated to include predication optimizations. Added the predicate no stall instructions,
Rev 2.04 (Laurent Lefebvre) Date :August 2, 2002	Documented the new parameter generation scheme for XY coordinates points and lines STs.
Rev 2.05 (Laurent Lefebvre) Date : September 10, 2002	Some interface changes and an architectural change to the auto-counter scheme.
Rev 2.06 (Laurent Lefebvre) Date : October 11, 2002	Widened the event interface to 5 bits. Some other little typos corrected.
Rev 2.07 (Laurent Lefebvre) Date : October 14, 2002	Loops, jumps and calls are now using a 13 bit address which allows to jump and call and loop around any control flow addresses (does not requires to be even anymore).
Rev 2.08 (Laurent Lefebvre) Date : October 16, 2002	Clarification updates after discussion with Clay.
Rev 2.09 (Laurent Lefebvre) Date : January 7, 2003	Corrected the SQ→SP staging register interface.

1. Overview

The sequencer chooses two ALU threads and a fetch hread to execute, and executes all of the instructions in a block before looking for a new clause of the same type. Two ALU threads are executed interleaved to hide the ALU latency. The arbitrator will give priority to older threads. There are two separate reservation stations, one for pixel vectors and one for vertices vectors. This way a pixel can pass a vertex and a vertex can pass a pixel.

To support the shader pipe the sequencer also contains the shader instruction cache, constant store, control flow constants and texture state. The four shader pipes also execute the same instruction thus there is only one sequencer for the whole chip.

The sequencer first arbitrates between vectors of 64 vertices that arrive directly from primitive assembly and vectors of 16 quads (64 pixels) that are generated in the scan converter.

The vertex or pixel program specifies how many GPRs it needs to execute. The sequencer will not start the next vector until the needed space is available in the GPRs.

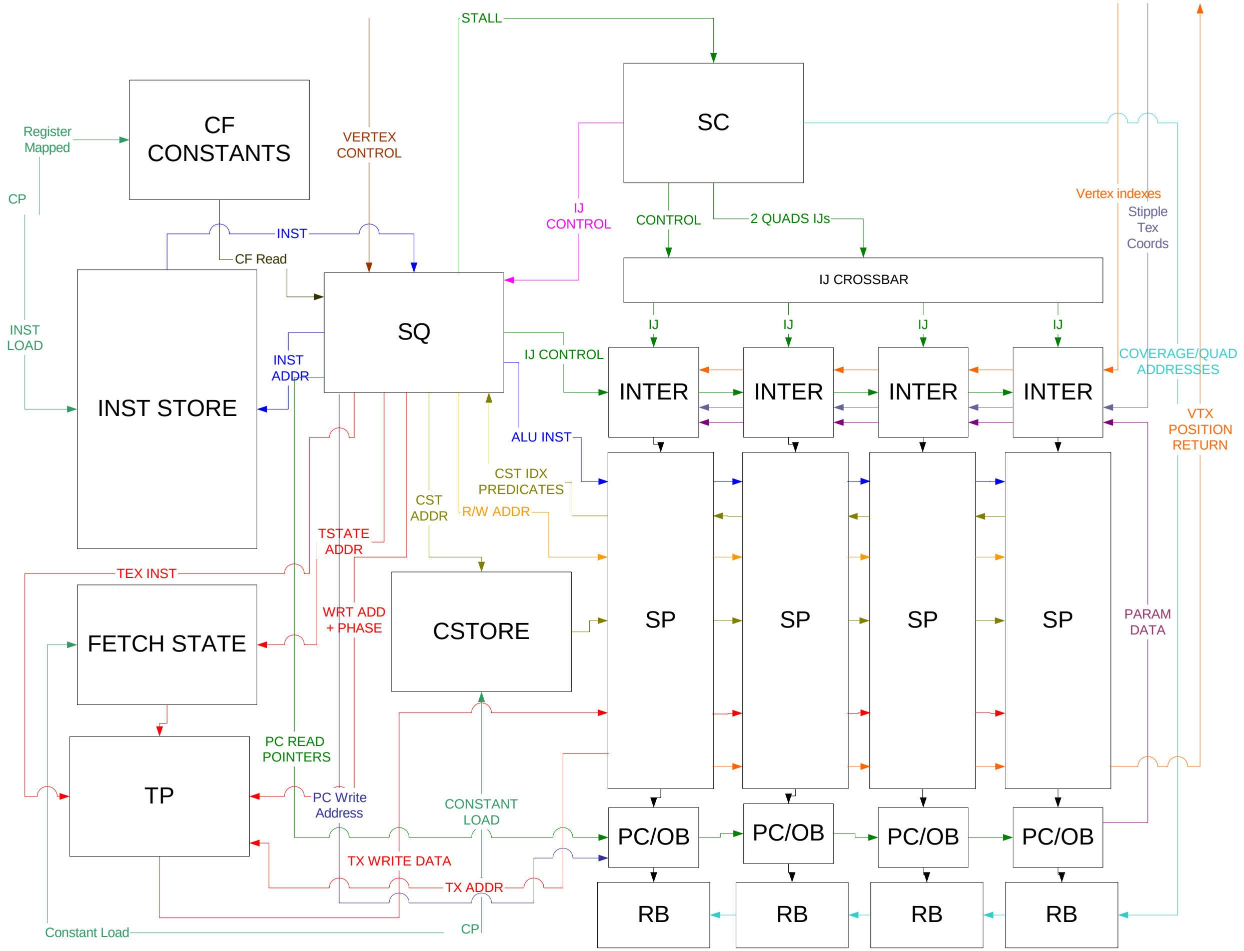


Figure 1: General Sequencer overview

1.1 Top Level Block Diagram

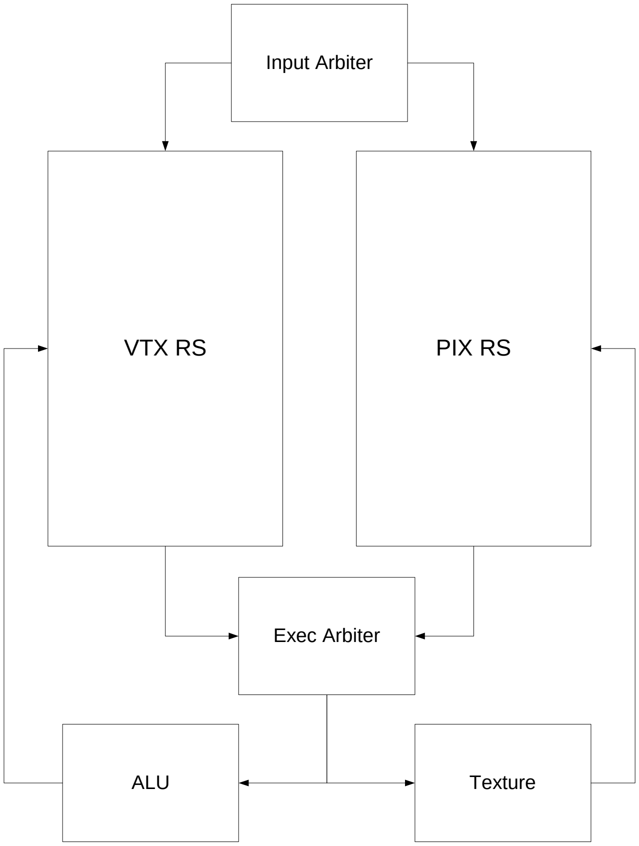


Figure 2: Reservation stations and arbiters

Under this new scheme, the sequencer (SQ) will only use one global state management machine per vector type (pixel, vertex) that we call the reservation station (RS).

1.2 Data Flow graph (SP)

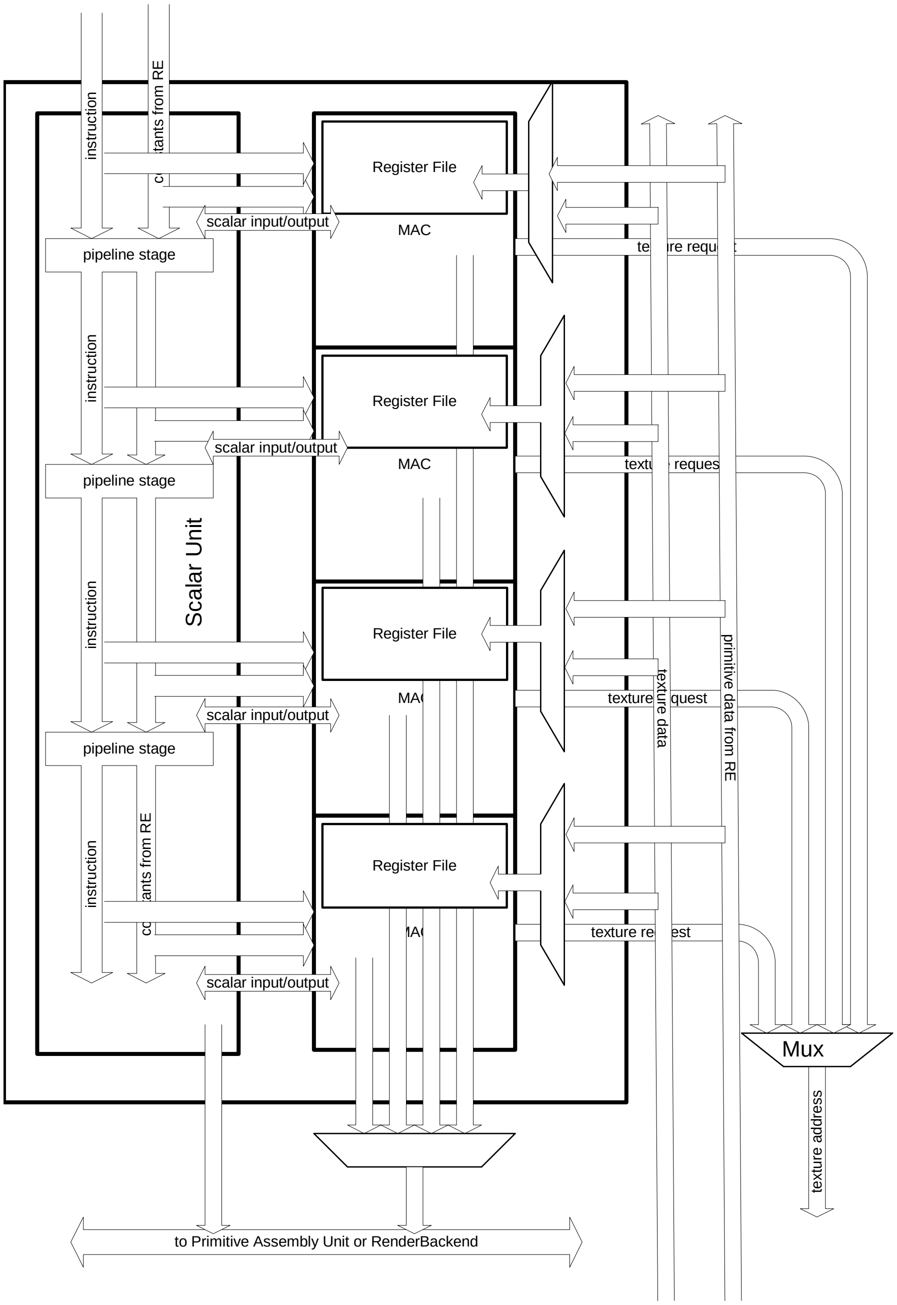


Figure 3: The shader Pipe

The gray area represents blocks that are replicated 4 times per shader pipe (16 times on the overall chip).

1.3 Control Graph

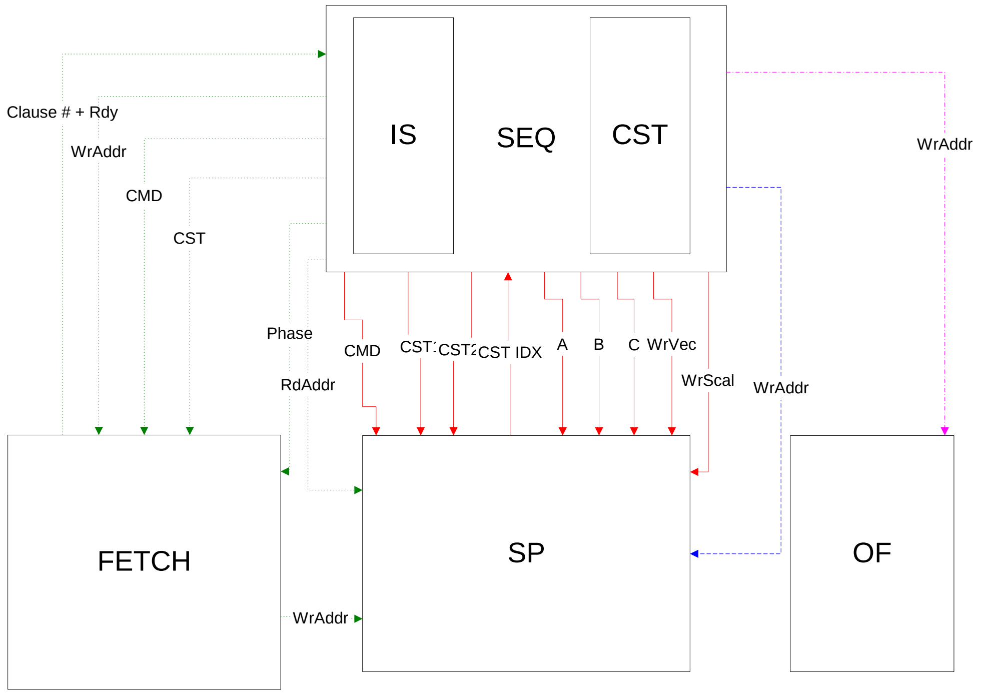


Figure 4: Sequencer Control interfaces

In green is represented the Fetch control interface, in red the ALU control interface, in blue the Interpolated/Vector control interface and in purple is the output file control interface.

2. Interpolated data bus

The interpolators contain an IJ buffer to pack the information as much as possible before writing it to the register file.

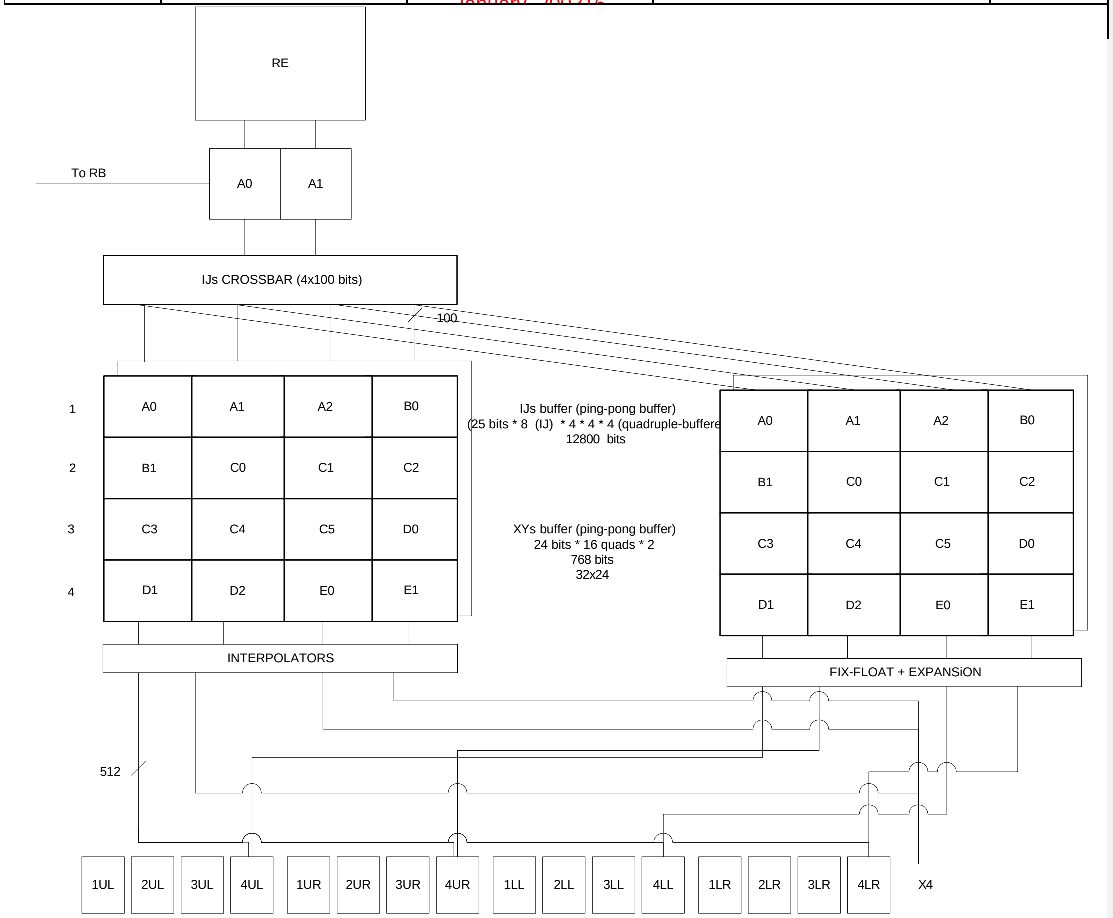


Figure 5: Interpolation buffers

WRITES

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21	T22	T23
SP 0	A0	A0	XY A0	B1	B1	XY B1	C3	C3	XY C3				D1	D1	XY D1									
SP 1	A1	A1	XY A1				C0	C0	XY C0	C4	C4	XY C4	D2	D2	XY D2									
SP 2	A2	A2	XY A2				C1	C1	XY C1	C5	C5	XY C5				E0	E0	XY E0						
SP 3				B0	B0	XY B0	C2	C2	XY C2				D0	D0	XY D0	E1	E1	XY E1						

READS

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21	T22	T23
SP 0	XY 0-3	XY 16-19	XY 32-35	XY 48-51	A0	B1	C3	D1					A0	B1	C3	D1					V 0-3	V 16-19	V 32-35	V 48-51
SP 1	XY 4-7	XY 20-23	XY 36-39	XY 52-55	A1		C4	D2		C0			A1		C4	D2		C0			V 4-7	V 20-23	V 36-39	V 52-55
SP 2	XY 8-11	XY 24-27	XY 40-43	XY 56-59	A2		C5			C1		E0	A2		C5			C1		E0	V 8-11	V 24-27	V 40-43	V 56-59
SP 3	XY 12-15	XY 28-31	XY 44-47	XY 60-63					B0	C2	D0	E1					B0	C2	D0	E1	V 12-15	V 28-31	V 44-47	V 60-63
	XY				P1								P2								VTX			

Figure 6: Interpolation timing diagram

Above is an example of a tile the sequencer might receive from the SC. The write side is how the data get stacked into the XY and IJ buffers, the read side is how the data is passed to the GPRs. The IJ information is packed in the IJ buffer 4 quads at a time or two clocks. The sequencer allows at any given time as many as four quads to interpolate a parameter. They all have to come from the same primitive. Then the sequencer controls the write mask to the GPRs to write the valid data in.

3. Instruction Store

There is going to be only one instruction store for the whole chip. It will contain 4096 instructions of 96 bits each.

It is likely to be a 1 port memory; we use 1 clock to load the ALU instruction, 1 clocks to load the Fetch instruction, 1 clock to load 2 control flow instructions and 1 clock to write instructions.

The instruction store is loaded by the CP thru the register mapped registers.

The VS_BASE and PS_BASE context registers are used to specify for each context where its shader is in the instruction memory.

For the Real time commands the story is quite the same but for some small differences. There are no wrap-around points for real time so the driver must be careful not to overwrite regular shader data. The shared code (shared subroutines) uses the same path as real time.

4. Sequencer Instructions

All control flow instructions instructions are handled by the sequencer only. The ALUs will perform NOPs during this time (MOV PV,PV, PS,PS) if they have nothing else to do.

5. Constant Stores

5.1 Memory organizations

A likely size for the ALU constant store is 1024x128 bits. The read BW from the ALU constant store is 128 bits/clock and the write bandwidth is 32 bits/clock (directed by the CP bus size not by memory ports).

The maximum logical size of the constant store for a given shader is 256 constants. Or 512 for the pixel/vertex shader pair. The size of the re-mapping table is 128 lines (each line addresses 4 constants). The write granularity is 4 constants or 512 bits. It takes 16 clocks to write the four constants. Real time requires 256 lines in the physical memory (this is physically register mapped).

The texture state is also kept in a similar memory. The size of this memory is 320x96 bits (128 texture states for regular mode, 32 states for RT). The memory thus holds 128 texture states (192 bits per state). The logical size exposes 32 different states total, which are going to be shared between the pixel and the vertex shader. The size of the re-mapping table to for the texture state memory is 32 lines (each line addresses 1 texture state lines in the real memory). The CP write granularity is 1 texture state lines (or 192 bits). The driver sends 512 bits but the CP ignores the top 320 bits. It thus takes 6 clocks to write the texture state. Real time requires 32 lines in the physical memory (this is physically register mapped).

The control flow constant memory doesn't sit behind a renaming table. It is register mapped and thus the driver must reload its content each time there is a change in the control flow constants. Its size is 320*32 because it must hold 8 copies of the 32 dwords of control flow constants and the loop construct constants must be aligned.

The constant re-mapping tables for texture state and ALU constants are logically register mapped for regular mode and physically register mapped for RT operation.

5.2 Management of the Control Flow Constants

The control flow constants are register mapped, thus the CP writes to the according register to set the constant, the SQ decodes the address and writes to the block pointed by its current base pointer (CF_WR_BASE). On the read side, one level of indirection is used. A register (SQ_CONTEXT_MISC.CF_RD_BASE) keeps the current base pointer to the control flow block. This register is copied whenever there is a state change. Should the CP write to CF after the state change, the base register is updated with the (current pointer number +1)% number of states. This way, if the CP doesn't write to CF the state is going to use the previous CF constants.

5.3 Management of the re-mapping tables

5.3.1 *R400 Constant management*

The sequencer is responsible to manage two re-mapping tables (one for the constant store and one for the texture state). On a state change (by the driver), the sequencer will broadside copy the contents of its re-mapping tables to a new one. We have 8 different re-mapping tables we can use concurrently.

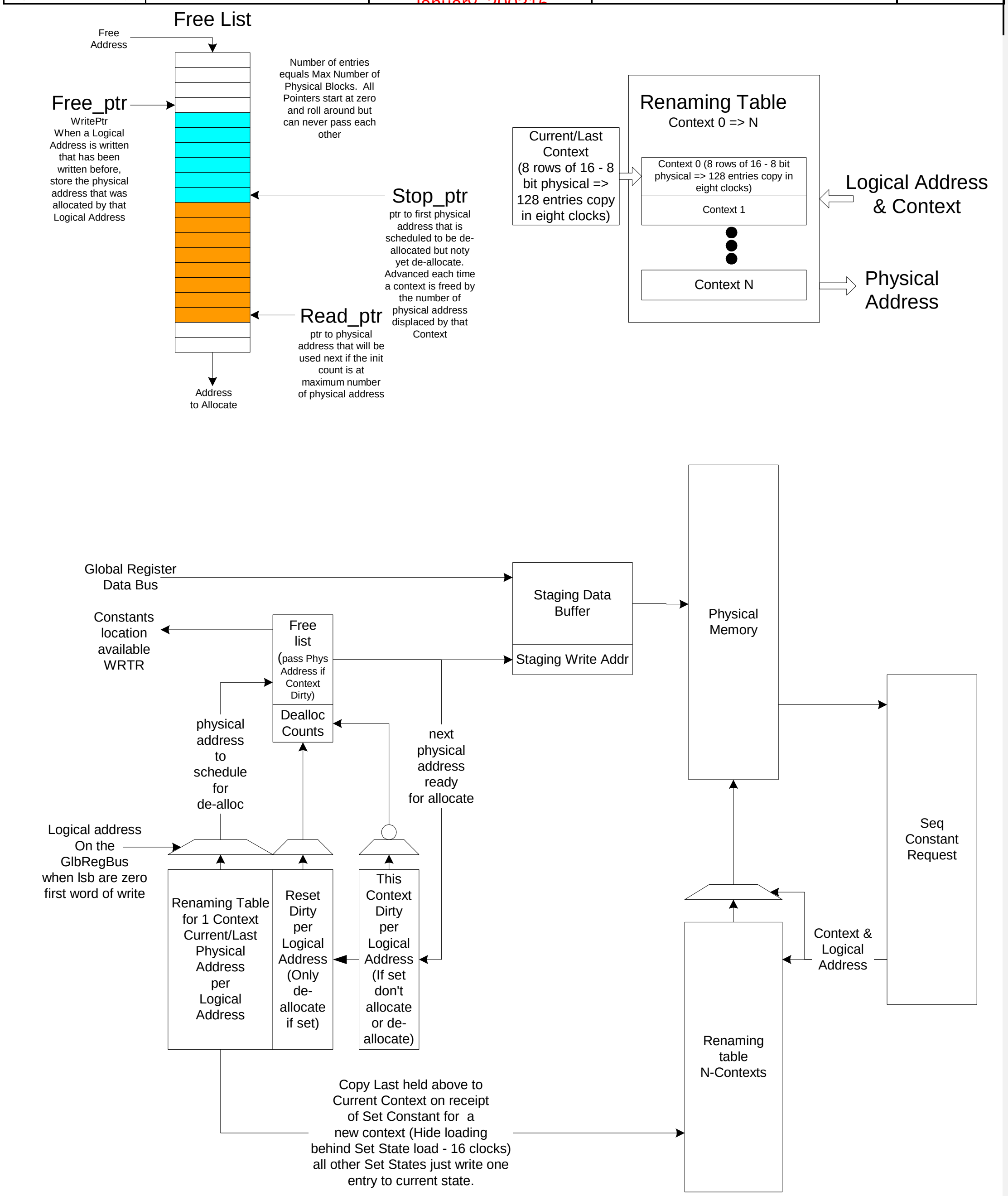
The constant memory update will be incremental, the driver only need to update the constants that actually changed between the two state changes.

For this model to work in its simplest form, the requirement is that the physical memory MUST be at least twice as large as the logical address space + the space allocated for Real Time. In our case, since the logical address space is 512 and the reserved RT space can be up to 256 entries, the memory must be of sizes 1280 and above. Similarly the size of the texture store must be of 32*2+32 = 96 entries and above.

5.3.2 *Proposal for R400LE constant management*

To make this scheme work with only 512+256 = 768 entries, upon reception of a CONTROL packet of state + 1, the sequencer would check for SQ_IDLE and PA_IDLE and if both are idle will erase the content of state to replace it with the new state (this is depicted in Figure 8: De-allocation mechanism~~Figure 8: De-allocation mechanismFigure 8: De-allocation mechanism~~). Note that in the case a state is cleared a value of 0 is written to the corresponding de-allocation counter location so that when the SQ is going to report a state change, nothing will be de-allocated upon the first report.

The second path sets all context dirty bits that were used in the current state to 1 (thus allowing the new state to reuse these physical addresses if needed).



Free List
Free Address
Free_ptr
WritePtr
When a Logical Address is written that has been written before, store the physical address that was allocated by that Logical Address
Number of entries equals Max Number of Physical Blocks. All Pointers start at zero and roll around but can never pass each other
Stop_ptr
ptr to first physical address that is scheduled to be de-allocated but noty yet de-allocate. Advanced each time a context is freed by the number of physical address displaced by that Context
Read_ptr
ptr to physical address that will be used next if the init count is at maximum number of physical address
Address to Allocate
Current/Last Context (8 rows of 16 - 8 bit physical => 128 entries copy in eight clocks)
Renaming Table
Context 0 => N
Context 0 (8 rows of 16 - 8 bit physical => 128 entries copy in eight clocks)
Context 1
Context N
Logical Address & Context
Physical Address
Global Register Data Bus
Staging Data Buffer
Staging Write Addr
Physical Memory
Constants location available WRTR
Free list (pass Phys Address if Context Dirty)
Dealloc Counts
physical address to schedule for de-alloc
next physical address ready for allocate
Logical address On the GlbRegBus when lsb are zero first word of write
Renaming Table for 1 Context Current/Last Physical Address per Logical Address
Reset Dirty per Logical Address (Only de-allocate if set)
This Context Dirty per Logical Address (If set don't allocate or de-allocate)
Seq Constant Request
Context & Logical Address
Renaming table N-Contexts
Copy Last held above to Current Context on receipt of Set Constant for a new context (Hide loading behind Set State load - 16 clocks) all other Set States just write one entry to current state.

Figure 7: Constant management

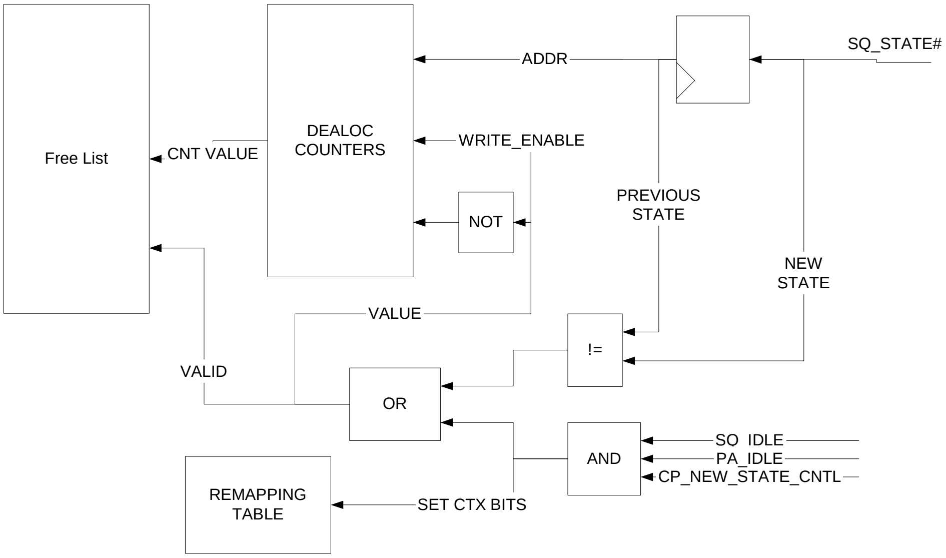


Figure 8: De-allocation mechanism for R400LE

5.3.3 *Dirty bits*

Two sets of dirty bits will be maintained per logical address. The first one will be set to zero on reset and set when the logical address is addressed. The second one will be set to zero whenever a new context is written and set for each address written while in this context. The reset dirty is not set, then writing to that logical address will not require de-allocation of whatever address stored in the renaming table. If it is set and the context dirty is not set, then the physical address store needs to be de-allocated and a new physical address is necessary to store the incoming data. If they are both set, then the data will be written into the physical address held in the renaming for the current logical address. No de-allocation or allocation takes place. This will happen when the driver does a set constant twice to the same logical address between context changes. NOTE: It is important to detect and prevent this, failure to do it will allow multiple writes to allocate all physical memory and thus hang because a context will not fit for rendering to start and thus free up space.

5.3.4 *Free List Block*

A free list block that would consist of a counter (called the IFC or Initial Free Counter) that would reset to zero and incremented every time a chunk of physical memory is used until they have all been used once. This counter would be checked each time a physical block is needed, and if the original ones have not been used up, us a new one, else check the free list for an available physical block address. The count is the physical address for when getting a chunk from the counter.

Storage of a free list big enough to store all physical block addresses.

Maintain three pointers for the free list that are reset to zero. The first one we will call write_ptr. This pointer will identify the next location to write the physical address of a block to be de-allocated. Note: we can never free more physical memory locations than we have. Once recording address the pointer will be incremented to walk the free list like a ring.

The second pointer will be called stop_ptr. The stop_ptr pointer will be advanced by the number of address chunks de-allocates when a context finishes. The address between the stop_ptr and write_ptr cannot be reused because they are still in use. But as soon as the context using then is dismissed the stop_ptr will be advanced.

The third pointer will be called read_ptr. This pointer will point will point to the next address that can be used for allocation as long as the read_ptr does not equal the stop_ptr and the IFC is at its maximum count.

5.3.5 *De-allocate Block*

This block will maintain a free physical address block count for each context. While in current context, a count shall be maintained specifying how many blocks were written into the free list at the write_ptr pointer. This count will be reset upon reset or when this context is active on the back and different than the previous context. It is actually a count of blocks in the previous context that will no longer be used. This count will be used to advance the write_ptr pointer to make available the set of physical blocks freed when the previous context was done. This allows the discard or de-allocation of any number of blocks in one clock.

5.3.6 *Operation of Incremental model*

The basic operation of the model would start with the write_ptr, stop_ptr, read_ptr pointers in the free list set to zero and the free list counter is set to zero. Also all the dirty bits and the previous context will be initialized to zero. When the first set constants happen, the reset dirty bit will not be set, so we will allocate a physical location from the free list counter because its not at the max value. The data will be written into physical address zero. Both the additional copy of the renaming table and the context zeros of the big renaming table will be updated for the logical address that was written by set start with physical address of 0. This process will be repeated for any logical address that are not dirty until the context changes. If a logical address is hit that has its dirty bits set while in the same context, both dirty bits would be set, so the new data will be over-written to the last physical address assigned for this logical address. When the first draw command of the context is detected, the previous context stored in the additional renaming table will be copied to the larger renaming table in the current (new) context location. Then the set constant logical address with be loaded with a new physical address during the copy and if the reset dirty was set, the physical address it replaced in the renaming table would be entered at the write_ptr pointer location on the free list and the write_ptr will be incremented. The de-allocation counter for the previous context (eight) will be incremented. This as set states come in for this context one of the following will happen:

1.) No dirty bits are set for the logical address being updated. A line will be allocated of the free-list counter or the free list at read_ptr pointer if read_ptr != to stop_ptr .
2.) Reset dirty set and Context dirty not set. A new physical address is allocated, the physical address in the renaming table is put on the free list at write_ptr and it is incremented along with the de-allocate counter for the last context.
3.) Context dirty is set then the data will be written into the physical address specified by the logical address.

This process will continue as long as set states arrive. This block will provide backpressure to the CP whenever he has not free list entries available (counter at max and stop_ptr == read_ptr). The command stream will keep a count of contexts of constants in use and prevent more than max constants contexts from being sent.

Whenever a draw packet arrives, the content of the re-mapping table is written to the correct re-mapping table for the context number. Also if the next context uses less constants than the current one all exceeding lines are moved to the free list to be de-allocated later. This happens in parallel with the writing of the re-mapping table to the correct memory.

Now preferable when the constant context leaves the last ALU clause it will be sent to this block and compared with the previous context that left. (Init to zero) If they differ than the older context will no longer be referenced and thus can be de-allocated in the physical memory. This is accomplished by adding the number of blocks freed this context to the stop_ptr pointer. This will make all the physical addresses used by this context available to the read_ptr allocate pointer for future allocation.

This device allows representation of multiple contexts of constants data with N copies of the logical address space. It also allows the second context to be represented as the first set plus some new additional data by just storing the delta's. It allows memory to be efficiently used and when the constants updates are small it can store multiple context. However, if the updates are large, less contexts will be stored and potentially performance will be degraded. Although it will still perform as well as a ring could in this case.

5.4 Constant Store Indexing

In order to do constant store indexing, the sequencer must be loaded first with the indexes (that come from the GPRs). There are 144 wires from the exit of the SP to the sequencer (9 bits pointers x 16 vertexes/clock).

```
MOVA  R1.X,R2.X        // Loads the sequencer with the content of R2.X, also copies the content of R2.X into R1.X
ADD   R3,R4,C0[R2.X]// Uses the state from the sequencer to add R4 to C0[R2.X] into R3
```

Note that we don't really care about what is in the brackets because we use the state from the MOVA instruction. R2.X is just written again for the sake of simplicity and coherency.

The storage needed in the sequencer in order to support this feature is 2*64*9 bits = 1152 bits.

The address register is a signed integer, which ranges from –256 to 255.

5.5 Real Time Commands

The real time commands constants are written by the CP using the register mapped registers allocated for RT. It works is the same way than when dealing with regular constant loads BUT in this case the CP is not sending a logical address but rather a physical address and the reads are not passing thru the re-mapping table but are directly read from the memory. The boundary between the two zones is defined by the CONST_EO_RT control register. Similarly, for the fetch state, the boundary between the two zones is defined by the TSTATE_EO_RT control register.

5.6 Constant Waterfalling

In order to have a reasonable performance in the case of constant store indexing using the address register, we are going to have the possibility of using the physical memory port for read only. This way we can read 1 constant per clock and thus have a worst-case waterfall mode of 1 vertex per clock. There is a small synchronization issue related with this as we need for the SQ to make sure that the constants where actually written to memory (not only sent to the sequencer) before it can allow the first vector of pixels or vertices of the state to go thru the ALUs. To do so, the sequencer keeps 8 bits (one per render state) and sets the bits whenever the last render state is written to memory and clears the bit whenever a state is freed.

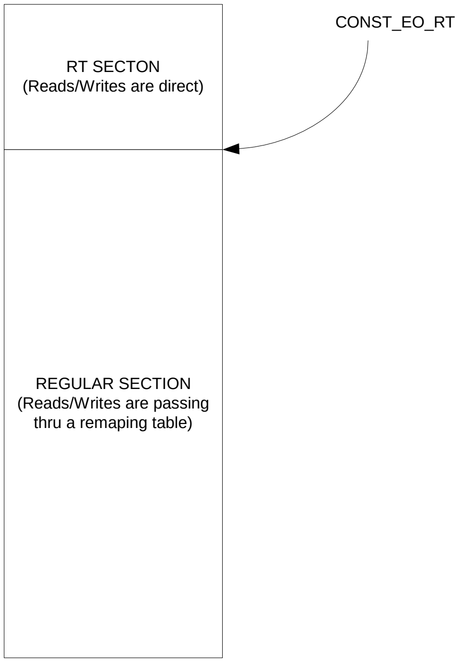


Figure 9: The Constant store

6. Looping and Branches

Loops and branches are planned to be supported and will have to be dealt with at the sequencer level. We plan on supporting constant loops and branches using a control program.

6.1 The controlling state.

The R400 controling state consists of:

Boolean[255:0]
Loop_count[7:0][31:0]
Loop_Start[7:0][31:0]
Loop_Step[7:0][31:0]

That is 256 Booleans and 32 loops.

We have a stack of 4 elements for nested calls of subroutines and 4 loop counters to allow for nested loops.

This state is available on a per shader program basis.

6.2 The Control Flow Program

We'd like to be able to code up a program of the form:

```
1:      Loop
2:      Exec    TexFetch
3:              TexFetch
4:              ALU
5:              ALU
6:              TexFetch
7:      End Loop
8:      ALU Export
```

But realize that 3: may be dependent on 2: and 4: is almost certainly dependent on 2: and 3:. Without clausing, these dependencies need to be expressed in the Control Flow instructions. Additionally, without separate 'texture clauses' and 'ALU clauses' we need to know which instructions to dispatch to the Texture Unit and which to the ALU unit. This information will be encapsulated in the flow control instructions.

Each control flow instruction will contain 2 bits of information for each (non-control flow) instruction:
a) ALU or Texture
b) Serialize Execution

(b) would force the thread to stop execution at this point (before the instruction is executed) and wait until all textures have been fetched. Given the allocation of reserved bits, this would mean that the count of an 'Exec' instruction would be limited to about 8 (non-control-flow) instructions. If more than this were needed, a second Exec (with the same conditions) would be issued.

Another function that relies upon 'clauses' is allocation and order of execution. We need to assure that pixels and vertices are exported in the correct order (even if not all execution is ordered) and that space in the output buffers are allocated in order. Additionally data can't be exported until space is allocated. A new control flow instruction:

Alloc <buffer select -- position,parameter, pixel or vertex memory. And the size required>.

would be created to mark where such allocation needs to be done. To assure allocation is done in order, the actual allocation for a given thread can not be performed unless the equivalent allocation for all previous threads is already completed. The implementation would also assure that execution of instruction(s) following the serialization due to the Alloc will occur in order -- at least until the next serialization or change from ALU to Texture. In most cases this will allow the exports to occur without any further synchronization. Only 'final' allocations or position allocations are

guaranteed to be ordered. Because strict ordering is required for pixels, parameters and positions, this implies only a single alloc for these structures. Vertex exports to memory do not require ordering during allocation and so multiple 'allocs' may be done.

6.2.1 *Control flow instructions table*

Here is the revised control flow instruction set.

Note that whenever a field is marked as RESERVED, it is assumed that all the bits of the field are cleared (0).

NOP		
47 … 44	43	42 … 0
0000	Addressing	RESERVED

This is a regular NOP.

Execute					
47 … 44	43	40 … 34	33 …16	15…12	11 … 0
0001	Addressing	RESERVED	Instructions type + serialize (9 instructions)	Count	Exec Address

Execute_End					
47 … 44	43	40 … 34	33 …16	15…12	11 … 0
0010	Addressing	RESERVED	Instructions type + serialize (9 instructions)	Count	Exec Address

Execute up to 9 instructions at the specified address in the instruction memory. The Instruction type field tells the sequencer the type of the instruction (LSB) (1 = Texture, 0 = ALU and whether to serialize or not the execution (MSB) (1 = Serialize, 0 = Non-Serialized). If Execute_End this is the last execution block of the shader program.

Conditional_Execute						
47 … 44	43	42	41 … 34	33…16	15 …12	11 … 0
0011	Addressing	Condition	Boolean address	Instructions type + serialize (9 instructions)	Count	Exec Address

Conditional_Execute_End						
47 … 44	43	42	41 … 34	33…16	15 …12	11 … 0
0100	Addressing	Condition	Boolean address	Instructions type + serialize (9 instructions)	Count	Exec Address

If the specified Boolean (8 bits can address 256 Booleans) meets the specified condition then execute the specified instructions (up to 9 instructions). If the condition is not met, we go on to the next control flow instruction. If Conditional_Execute_End and the condition is met, this is the last execution block of the shader program.

Conditional_Execute_Predicates							
47 … 44	43	42	41 … 36	35 … 34	33…16	15…12	11 … 0
0101	Addressing	Condition	RESERVED	Predicate vector	Instructions type + serialize (9 instructions)	Count	Exec Address

Conditional_Execute_Predicates_End							
47 … 44	43	42	41 … 36	35 … 34	33…16	15…12	11 … 0
0110	Addressing	Condition	RESERVED	Predicate vector	Instructions type + serialize (9 instructions)	Count	Exec Address

Check the AND/OR of all current predicate bits. If AND/OR matches the condition execute the specified number of instructions. We need to AND/OR this with the kill mask in order not to consider the pixels that aren't valid. If the

condition is not met, we go on to the next control flow instruction. If Conditional_Execute_Predicates_End and the condition is met, this is the last execution block of the shader program.

Conditional_Execute_Predicates_No_Stall							
47 … 44	43	42	41 … 36	35 … 34	33…16	15…12	11 … 0
1101	Addressing	Condition	RESERVED	Predicate vector	Instructions type + serialize (9 instructions)	Count	Exec Address

Conditional_Execute_Predicates_No_Stall_End							
47 … 44	43	42	41 … 36	35 … 34	33…16	15…12	11 … 0
1110	Addressing	Condition	RESERVED	Predicate vector	Instructions type + serialize (9 instructions)	Count	Exec Address

Same as Conditionnal_Execute_Predicates but the SQ is not going to wait for the predicate vector to be updated. You can only set this in the compiler if you know that the predicate set is only a refinement of the current one (like a nested if) because the optimization would still work.

Loop_Start					
47 … 44	43	42 … 21	20 … 16	15…13	12 … 0
0111	Addressing	RESERVED	loop ID	RESERVED	Jump address

Loop Start. Compares the loop iterator with the end value. If loop condition not met jump to the address. Forward jump only. Also computes the index value. The loop id must match between the start to end, and also indicates which control flow constants should be used with the loop.

Loop_End									
47 …44	43	42	41… 36	35…34	33… 22	21	20 … 16	15…13	12 … 0
1000	Addressing	Cond	RESERVED	Predicate Vector	RESERVED	Pred break	loop ID	RESERVED	start address

Loop end. Increments the counter by one, compares the loop count with the end value. If loop condition met, continue, else, jump BACK to the start of the loop. If predicate break != 0, then compares predicate vector n (specified by predicate Vector) to condition. If all bits meet condition then break the loop.

The way this is described does not prevent nested loops, and the inclusion of the loop id make this easy to do.

Conditionnal_Call						
47 … 44	43	42	41 … 34	33 … 14	13	12 … 0
1001	Addressing	Condition	Boolean address	RESERVED	Force Call	Jump address

If the condition is met, jumps to the specified address and pushes the control flow program counter on the stack. If force call is set the condition is ignored and the call is made always.

Return		
47 … 44	43	42 … 0
1010	Addressing	RESERVED

Pops the topmost address from the stack and jumps to that address. If nothing is on the stack, the program will just continue to the next instruction.

Conditionnal_Jump							
47 … 44	43	42	41… 34	33	32 … 14	13	12 … 0
1011	Addressing	Condition	Boolean address	FW only	RESERVED	Force Jump	Jump address

If force jump is set the condition is ignored and the jump is made always. If FW only is set then only forward jumps are allowed.

Allocate				
47 … 44	43	42…41	40 … 3	2…0
1100	Debug	Buffer Select	RESERVED	Size

Buffer Select takes a value of the following:
01 – position export (ordered export)
10 – parameter cache or pixel export (ordered export)
11 – pass thru (out of order exports).

Size field is only used to reserve space in the export buffer for pass thru exports. Valid values are 1 (1 line) thru 9 (9 lines). It should be determined by the compiler/assembler by taking max index used +1.

If debug is set this is a debug alloc (ignore if debug DB_ON register is set to off).

6.3 Implementation

The envisioned implementation has a buffer that maintains the state of each thread. A thread lives in a given location in the buffer during its entire life, but the buffer has FIFO qualities in that threads leave in the order that they enter. Actually two buffers are maintained -- one for Vertices and one for Pixels. The intended implementation would allow for:

16 entries for vertices
48 entries for pixels.

From each buffer, arbitration logic attempts to select 1 thread for the texture unit and 2 (interleaved) thread for the ALU unit. Once a thread is selected it is read out of the buffer, marked as invalid, and submitted to appropriate execution unit. It is returned to the buffer (at the same place) with its status updated once all possible sequential instructions have been executed. A switch from ALU to TEX or visa-versa or a Serialize_Execution modifier forces the thread to be returned to the buffer.

Each entry in the buffer will be stored across two physical pieces of memory - most bits will be stored in a 1 read port device. Only bits needed for thread arbitration will be stored in a highly multi-ported structure. The bits kept in the 1 read port device will be termed 'state'. The bits kept in the multi-read ported device will be termed 'status'.

'State Bits' needed include:

1. Control Flow Instruction Pointer (13 bits),
2. Execution Count Marker 4 bits),
3. Loop Iterators (4x9 bits),
4. Loop Counters (4x9 bits),
5. Call return pointers (4x13 bits),
6. Predicate Bits (64 bits),
7. Export ID (1 bit),
8. Parameter Cache base Ptr (7 bits),
9. GPR Base Ptr (8 bits),
10. Context Ptr (3 bits).
11. LOD corrections (6x16 bits)
12. Valid bits (64 bits)
13. RT (1 bit) Signifies that this thread is a Real Time thread. This bit must be sent to the Constant store state machine when reading it.

Absent from this list are 'Index' pointers. These are costly enough that I'm presuming that they are instead stored in the GPRs. The first seven fields above (Control Flow Ptr, Execution Count, Loop Counts, call return ptrs, Predicate bits, PC base ptr and export ID) are updated every time the thread is returned to the buffer based on how much progress has been mode on thread execution. GPR Base Ptr, Context Ptr and LOD corrections are unchanged throughout execution of the thread.

'Status Bits' needed include:

- Valid Thread
- Texture/ALU engine needed
- Texture Reads are outstanding
- Waiting on Texture Read to Complete
- Allocation Wait (2 bits)
- 00 – No allocation needed
- 01 – Position export allocation needed (ordered export)
- 10 – Parameter or pixel export needed (ordered export)
- 11 – pass thru (out of order export)
- Allocation Size (4 bits)
- Position Allocated
- Mem/Color Allocated
- First thread of a new context
- Event thread (NULL thread that needs to trickle down the pipe)
- Last (1 bit)
- Pulse SX (1 bit)

All of the above fields from all of the entries go into the arbitration circuitry. The arbitration circuitry will select a winner for both the Texture Engine and for the ALU engine. There are actually two sets of arbitration -- one for pixels and one for vertices. A final selection is then done between the two. But the rest of this implementation summary only considers the 'first' level selection which is similar for both pixels and vertices.

Texture arbitration requires no allocation or ordering so it is purely based on selecting the 'oldest' thread that requires the Texture Engine.

ALU arbitration is a little more complicated. First, only threads where either of Texture_Reads_outstanding or Waiting_on_Texture_Read_to_Complete are '0' are considered. Then if Allocation_Wait is active, these threads are further filtered based on whether space is available. If the allocation is position allocation, then the thread is only considered if all 'older' threads have already done their position allocation (position allocated bits set). If the allocation is parameter or pixel allocation, then the thread is only considered if it is the oldest thread. Also a thread is not considered if it is a parameter or pixel or position allocation, has its First_thread_of_a_new_context bit set and would cause ALU interleaving with another thread performing the same parameter or pixel or position allocation. Finally the 'oldest' of the threads that pass through the above filters is selected. If the thread needed to allocate, then at this time the allocation is done, based on Allocation_Size. If a thread has its "last" bit set, then it is also removed from the buffer, never to return.

If I now redefine 'clauses' to mean 'how many times the thread is removed from the thread buffer for the purpose of exection by either the ALU or Texture engine', then the minimum number of clauses needed is 2 -- one to perform the allocation for exports (execution automatically halts after an 'Alloc' instruction) (but doesn't performs the actual allocation) and one for the actual ALU/export instructions. As the 'Alloc' instruction could be part of a texture clause (presumably the final instruction in such a clause), a thread could still execute in this minimal number of 2 clauses, even if it involved texture fetching.

The Texture_Reads_Outstanding bit must be updated by the sequencer, based on keeping track of how many Texture Clauses have been executed by a given thread that have not yet had there data returned. Any number above 0 results in this bit being set. We could consider forcing synchronization such that two texture clauses for a given thread may not be outstanding at any time (that would be my preference for simplicity reasons and because it would require only very little change in the texture pipe interface). This would allow the sequencer to set the bit on execution of the texture clause, and allow the texture unit to return a pointer to the thread buffer on completion that clears the bit.

6.4 Data dependant predicate instructions

Data dependant conditionals will be supported in the R400. The only way we plan to support those is by supporting three vector/scalar predicate operations of the form:

PRED_SETE_PUSH - similar to SETE except that the result is 'exported' to the sequencer.
PRED_SETNE_PUSH - similar to SETNE except that the result is 'exported' to the sequencer.
PRED_SETGT_PUSH - similar to SETGT except that the result is 'exported' to the sequencer
PRED_SETGTE_PUSH - similar to SETGTE except that the result is 'exported' to the sequencer

For the scalar operations only we will also support the two following instructions:
PRED_SETE
PRED_SETNE
PRED_SETGT
PRED_SET_INV
PRED_SET_POP
PRED_SET_CLR
PRED_SET_RESTORE

Details about actual implementation of these opcodes are in the shader pipe architectural spec.

The export is a single bit - 1 or 0 that is sent using the same data path as the MOVA instruction. The sequencer will maintain 1 set of 64 bits predicate vectors (in fact 2 sets because we interleave two programs but only 1 will be exposed) and use it to control the write masking. This predicate is maintained across clause boundaries.

Then we have two conditional execute bits. The first bit is a conditional execute “on” bit and the second bit tells us if we execute on 1 or 0. For example, the instruction:

P0_ ADD_# R0,R1,R2

Is only going to write the result of the ADD into those GPRs whose predicate bit is 0. Alternatively, P1_ADD_# would only write the results to the GPRs whose predicate bit is set. The use of the P0 or P1 without precharging the sequencer with a PRED instruction is undefined.

6.5 HW Detection of PV,PS

Because of the control program, the compiler cannot detect statically dependant instructions. In the case of non-masked writes and subsequent reads the sequencer will insert uses of PV,PS as needed. This will be done by comparing the read address and the write address of consecutive instructions. For masked writes, the sequencer will insert detect wich channels to read from the GPRs and which ones to read from the PV/PS.

6.6 Register file indexing

Because we can have loops in fetch clause, we need to be able to index into the register file in order to retrieve the data created in a fetch clause loop and use it into an ALU clause. The instruction will include the base address for register indexing and the instruction will contain these controls:

Bit7	Bit 6	
0	0	'absolute register'
0	1	'relative register'
1	0	'previous vector'
1	1	'previous scalar'

In the case of an absolute register we just take the address as is. In the case of a relative register read we take the base address and we add to it the loop_index and this becomes our new address that we give to the shader pipe.

The sequencer is going to keep a loop index computed as such:

Index = Loop_iterator*Loop_step + Loop_start.

We loop until loop_iterator = loop_count. Loop_step is a signed value [-128…127]. The computed index value is a 10 bit counter that is also signed. Its real range is [-256,256]. The tenth bit is only there so that we can provide an out of range value to the "indexing logic" so that it knows when the provided index is out of range and thus can make the necessary arrangements.

6.7 Debugging the Shaders

In order to be able to debug the pixel/vertex shaders efficiently, we provide 2 methods.

6.7.1 *Method 1: Debugging registers*

Current plans are to expose 2 debugging, or error notification, registers:
1. address register where the first error occurred
2. count of the number of errors

The sequencer will detect the following groups of errors:
- count overflow
- constant indexing overflow
- register indexing overflow

Compiler recognizable errors:
- jump errors
    relative jump address > size of the control flow program
- call stack
    call with stack full
    return with stack empty

With all the other errors, program can continue to run, potentially to worst-case limits.

If indexing outside of the constant or the register range, causing an overflow error, the hardware is specified to return the value with an index of 0. This could be exploited to generate error tokens, by reserving and initializing the 0th register (or constant) for errors.

{ISSUE : Interrupt to the driver or not?}

6.7.2 *Method 2: Exporting the values in the GPRs*

1) The sequencer will have a debug active, count register and an address register for this mode.

Under the normal mode execution follows the normal course.

Under the debug mode it is assumed that the program is always exporting n debug vectors and that all other exports to the SX block (but for position) will be turned off (changed into NOPs) by the sequencer (even if they occur before the address stated by the ADDR debug register).

7. Pixel Kill Mask

A vector of 64 bits is kept by the sequencer per group of pixels/vertices. Its purpose is to optimize the texture fetch requests and allow the shader pipe to kill pixels using the following instructions:

MASK_SETE
MASK_SETNE
MASK_SETGT
MASK_SETGTE

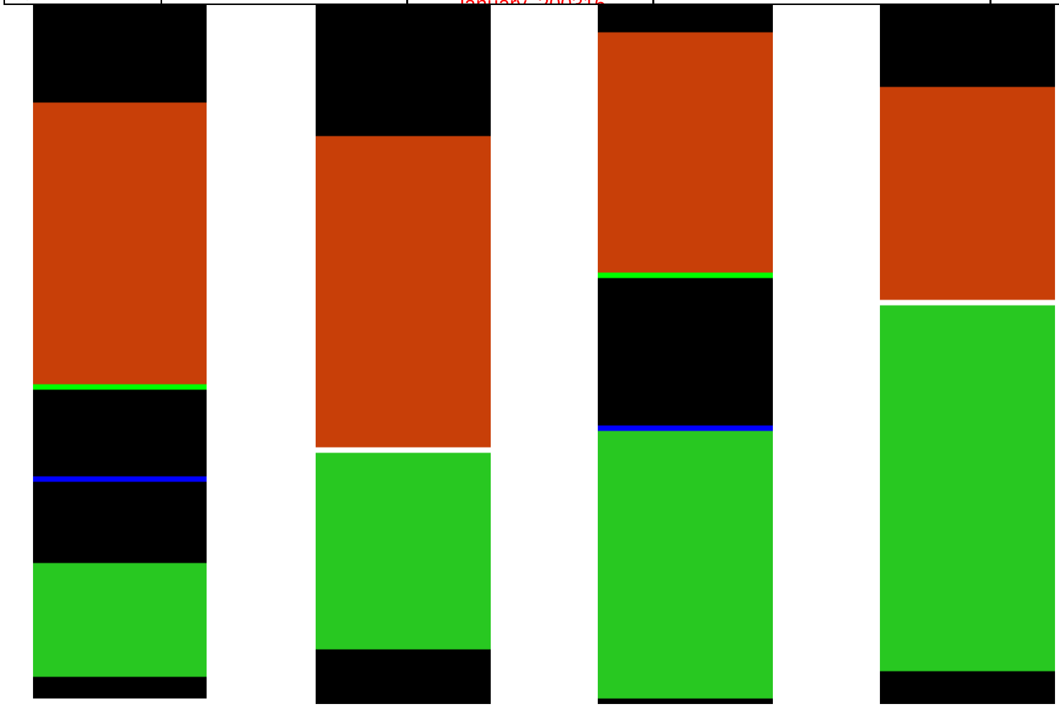
~~Multipass vertex shaders (HOS)~~

~~Multipass vertex shaders are able to export from the 6 last clauses but to memory ONLY.~~

~~9.~~8. Register file allocation

Formatted: Bullets and Numbering

The register file allocation for vertices and pixels can either be static or dynamic. In both cases, the register file in managed using two round robins (one for pixels and one for vertices). In the dynamic case the boundary between pixels and vertices is allowed to move, in the static case it is fixed to 128-VERTEX_REG_SIZE for vertices and PIXEL_REG_SIZE for pixels.



Above is an example of how the algorithm works. Vertices come in from top to bottom; pixels come in from bottom to top. Vertices are in orange and pixels in green. The blue line is the tail of the vertices and the green line is the tail of the pixels. Thus anything between the two lines is shared. When pixels meets vertices the line turns white and the boundary is static until both vertices and pixels share the same “unallocated bubble”. Then the boundary is allowed to move again. The numbering of the GPRs starts from the bottom of the picture at index 0 and goes up to the top at index 127.

Formatted: Bullets and Numbering

~~10.~~9. Fetch Arbitration

The fetch arbitration logic chooses one of the n potentially pending fetch clauses to be executed. The choice is made by looking at the Vs and Ps reservation stations and picking the first one ready to execute. Once chosen, the clause state machine will send one 2x2 fetch per clock (or 4 fetches in one clock every 4 clocks) until all the fetch instructions of the clause are sent. This means that there cannot be any dependencies between two fetches of the same clause.

The arbitrator will not wait for the fetches to return prior to selecting another clause for execution. The fetch pipe will be able to handle up to X(?) in flight fetches and thus there can be a fair number of active clauses waiting for their fetch return data.

Formatted: Bullets and Numbering

~~11.~~10. ALU Arbitration

ALU arbitration proceeds in almost the same way than fetch arbitration. The ALU arbitration logic chooses one of the n potentially pending ALU clauses to be executed. The choice is made by looking at the Vs and Ps reservation stations and picking the first one ready to execute. There are two ALU arbiters, one for the even clocks and one for the odd clocks. For example, here is the sequencing of two interleaved ALU clauses (E and O stands for Even and Odd sets of 4 clocks):

Einst0 Oinst0 Einst1 Oinst1 Einst2 Oinst2 Einst0 Oinst3 Einst1 Oinst4 Einst2 Oinst0…

Proceeding this way hides the latency of 8 clocks of the ALUs. Also note that the interleaving also occurs across clause boundaries.

~~12.~~11. Handling Stalls

Formatted: Bullets and Numbering

When the output file is full, the sequencer prevents the ALU arbitration logic from selecting the last clause (this way nothing can exit the shader pipe until there is place in the output file. If the packet is a vertex packet and the position buffer is full (POS_FULL) then the sequencer also prevents a thread from entering an exporting clause. The sequencer will set the OUT_FILE_FULL signal n clocks before the output file is actually full and thus the ALU arbiter will be able read this signal and act accordingly by not preventing exporting clauses to proceed.

~~13.~~12. Content of the reservation station FIFOs

Formatted: Bullets and Numbering

The reservation FIFOs contain the state of the vector of pixels and vertices. We have two sets of those: one for pixels, and one for vertices. They contain 3 bits of Render State 7 bits for the base address of the GPRs, some bits for LOD correction and coverage mask information in order to fetch fetch for only valid pixels, the quad address.

~~14.~~13. The Output File

Formatted: Bullets and Numbering

The output file is where pixels are put before they go to the RBs. The write BW to this store is 256 bits/clock. Just before this output file are staging registers with write BW 512 bits/clock and read BW 256 bits/clock. The staging registers are 4x128 (and there are 16 of those on the whole chip).

~~15.~~14. IJ Format

Formatted: Bullets and Numbering

The IJ information sent by the PA is of this format on a per quad basis:

We have a vector of IJ's (one IJ per pixel at the centroid of the fragment or at the center of the pixel depending on the mode bit). All pixel's parameters are always interpolated at full 20x24 mantissa precision.

$$P0 = A + I(0) * (B - A) + J(0) * (C - A)$$
$$P1 = A + I(1) * (B - A) + J(1) * (C - A)$$
$$P2 = A + I(2) * (B - A) + J(2) * (C - A)$$
$$P3 = A + I(3) * (B - A) + J(3) * (C - A)$$

P0	P1
P2	P3

Multiplies (Full Precision): 8
Subtracts 19x24 (Parameters): 2
Adds: 8

FORMAT OF P's IJ : Mantissa 20 Exp 4 for I + Sign
Mantissa 20 Exp 4 for J + Sign

Total number of bits : 20*8 + 4*8 + 4*2 = 200.

All numbers are kept using the un-normalized floating point convention: if exponent is different than 0 the number is normalized if not, then the number is un-normalized. The maximum range for the IJs (Full precision) is +/- 1024.

~~15.1~~14.1 Interpolation of constant attributes

Formatted: Bullets and Numbering

Because of the floating point imprecision, we need to take special provisions if all the interpolated terms are the same or if two of the terms are the same.

Formatted: Bullets and Numbering

~~16.~~15. Staging Registers

In order for the reuse of the vertices to be 14, the sequencer will have to re-order the data sent IN ORDER by the VGT for it to be aligned with the parameter cache memory arrangement. Given the following group of vertices sent by the VGT:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 || 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 || 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 || 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63

The sequencer will re-arrange them in this fashion:

0 1 2 3 16 17 18 19 32 33 34 35 48 49 50 51 || 4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55 || 8 9 10 11 24 25 26 27 40 41 42 43 56 57 58 59 || 12 13 14 15 28 29 30 31 44 45 46 47 60 61 62 63

The || markers show the SP divisions. In the event a shader pipe is broken, the SQ is responsible to insert padding to account for the missing pipe. For example, if SP1 is broken, vertices 4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55 will not be sent by the VGT to the SQ **AND** the SQ is responsible to “jump” over these vertices in order for no valid vertices to be sent to an invalid SP.

The most straightforward, *non-compressed* interface method would be to convert, in the VGT, the data to 32-bit floating point prior to transmission to the VSISRs. In this scenario, the data would be transmitted to (and stored in) the VSISRs in full 32-bit floating point. This method requires three 24-bit fixed-to-float converters in the VGT. Unfortunately, it also requires and additional 3,072 bits of storage across the VSISRs. This interface is illustrated in Figure 11~~Figure 11Figure 11~~. The area of the fixed-to-float converters and the VSISRs for this method is roughly estimated as 0.759sqmm using the R300 process. The gate count estimate is shown in Figure 10~~Figure 10Figure 10~~.

Basis for 8-deep Latch Memory (from R300)			
8x24-bit		11631 μ^2	60.57813 μ^2 per bit
Area of 96x8-deep Latch Memory		46524 μ^2	
Area of 24-bit Fix-to-float Converter		4712 μ^2 per converter	
Method 1	Block	Quantity	Area
	F2F	3	14136
	8x96 Latch	16	744384
			758520 μ^2

Figure 10:Area Estimate for VGT to Shader Interface

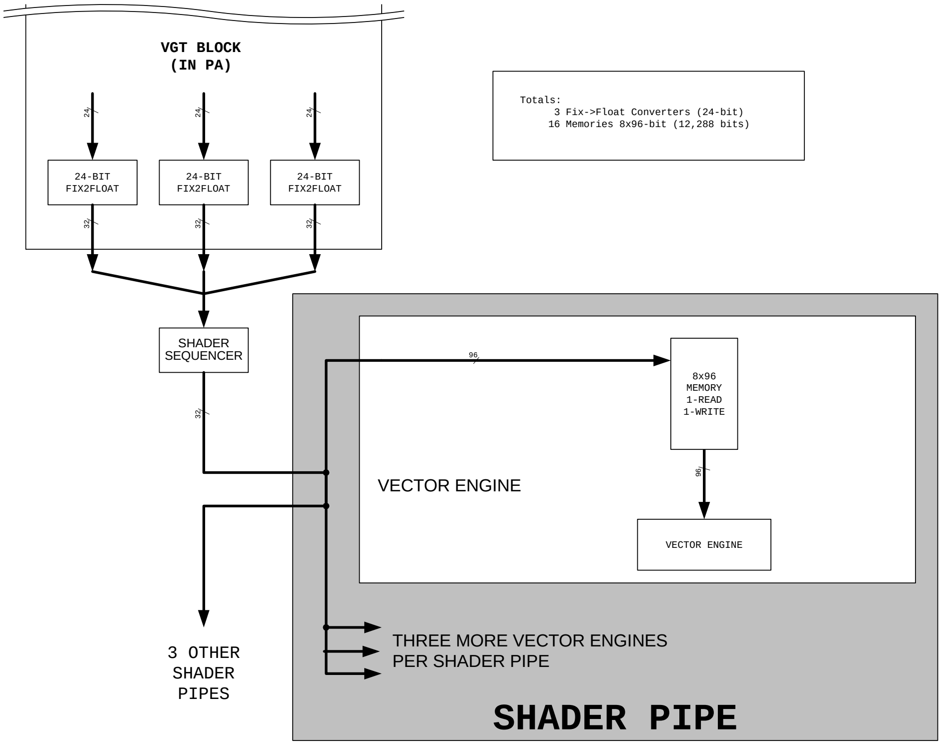


Figure 11:VGT to Shader Interface

Formatted: Bullets and Numbering

~~17.~~16. The parameter cache

The parameter cache is where the vertex shaders export their data. It consists of 16 128x128 memories (1R/1W). The reuse engine will make it so that all vertexes of a given primitive will hit different memories. The allocation method for these memories is a simple round robin. The parameter cache pointers are mapped in the following way: 4MSBs are the memory number and the 7 LSBs are the address within this memory.

MEMORY NUMBER 4 bits	ADDRESS 7 bits

The PA generates the parameter cache addresses as the positions come from the SQ. All it needs to do is keep a Current_Location pointer (7 bits only) and as the positions comes increment the memory number. When the memory number field wraps around, the PA increments the Current_Location by VS_EXPORT_COUNT (a snooped register from the SQ). As an example, say the memories are all empty to begin with and the vertex shader is exporting 8 parameters per vertex (VS_EXPORT_COUNT = 8). The first position received is going to have the PC address 00000000000 the second one 00010000000, third one 00100000000 and so on up to 11110000000. Then the next position received (the 17th) is going to have the address 00000001000, the 18th 00010001000, the 19th 00100001000 and so on. The Current_location is NEVER reset BUT on chip resets. The only thing to be careful about is that if the SX doesn't send you a full group of positions (<64) then you need to fill the address space so that the next group starts correctly aligned (for example if you receive only 33 positions then you need to add 2*VS_EXPORT_COUNT to Current_Location and reset the memory count to 0 before the next vector begins).

~~17.1~~16.1 Export restrictions

Formatted: Bullets and Numbering

~~17.1.1~~16.1.1 *Pixel exports:*

Pixels can export 1,2,3 or 4 color buffers to the SX(+z). The exports will be done in order. The exports will always be ordered to the SX.

Formatted: Bullets and Numbering

~~17.1.2~~16.1.2 *Vertex exports:*

Position or parameter caches can be exported in any order in the shader program. It is always better to export posistion as soon as possible. Position has to be exported in a single export block (no texture instructions can be placed between the exports). Parameter cache exports can be done in any order with texture instructions interleaved. The exports will always be allocated in order to the SX.

Formatted: Bullets and Numbering

~~17.1.3~~16.1.3 *Pass thru exports:*

Pass thru exports have to be done in groups of the form:

```
Alloc ~~4~~ 1 thru 5 (max export offset + 1, for example if using EM4 alloc size 5)~~(or 8)~~
Execute ALU(ADDR) ALU(DATA) ALU(DATA) ALU(DATA)…
```

When exporting to more than EM0, one MUST write to EM4 also (the write may be predicated if you don’t need the export). This is used to initialize the buffers in the SX.

There cannot be any serialize bits set OR texture Reads between the EA and the last EM.

Formatted

Memory exports will be surfaced using a macro extension; here is what needs to happen inside the macro:

The macro needs to create a special constant of the form:

Stream ID constant:
.x = Integer that holds BaseAddressInBytes/4 in bits (29:0). Bits 31:30 should be 0b01.
.y = 2**23
.z = Integer that holds register field data. Note that this data must be organized so that it always represents a 'valid' floating point number, with the relevant bits in (23 - 0); One way of doing this would be to take the 23 bits and add 2**23.
.w = max index value + 2**23

Output to EXaddress:

.x = Base of array (in low 30 bits)/4
.y = Index value (in low 23 bits)
.z = Register Field data (in low 23 bits)
.w = Max Index value (in low 23 bits)

Also Assume that C0:

.x = 0.0
.y = 1.0

The Macro expansion would be as follows:

MULADD EA = Rindex.xxxx,C0.xyxx,CstreamID;
MOV EMx (x = 0 thru 4) = Rdata;

The SX will check for invalid writes and **mask out the data** so it won’t be written to memory. Invalid writes are:

Formatted

1) Index value >= Max Index value
2) bit 31 != 0 (negative index)

Formatted: Bullets and Numbering

3) bits [30:23] != 23 + IEEE_EXP_BIAS (127) (meaning the index was too big to be represented using 23 bits)

They cannot have texture instructions interleaved in the export block. These exports **are not guaranteed to be ordered**.

Also, when doing a pass thru export~~, **Position MUST be exported AFTER all pass thru exports**. This position export is used to synchronize the chip when doing a transition from pass thru shader to regular shader and vice versa~~ the shader must still do either a position and PC export (if Vertex) or a color export (if Pixel). The pass thru export can occur anywhere in any shader program and thus can be used to debug. There can be any number of pass thru export blocks throughout the pixel or vertex shader or both.~~.~~

Formatted: Bullets and Numbering

~~17.2~~16.2 Arbitration restrictions

Here are the Sequencer arbitration restrictions:

1) Cannot execute a serialized thread if the corresponding texture pending bit is set
2) Cannot allocate position if any older thread has not allocated position
3) Cannot have more than 2 opened allocs of type : Memory, position and Color.
4) If last thread is marked as not valid AND marked as last and we are about to execute the second to oldest thread also marked last then:
    a. Both threads must be from the same context (cannot allow a first thread)
    b. Must turn off the predicate optimization for the second thread
5) Cannot execute a texture clause if texture reads are pending
6) Cannot execute last if texture pending (even if not serial)
7) Cannot allocate if not last or second to last for color exports.

Formatted: Bullets and Numbering

~~18.~~17. Export Types

The export type (or the location where the data should be put) is specified using the destination address field in the ALU instruction. Here is a list of all possible export modes:

Formatted: Bullets and Numbering

~~18.1~~17.1 Vertex Shading

0:15	- 16 parameter cache
16:31	- Empty (Reserved?)
32	- Export Address
33:37	- 5 vertex exports to the frame buffer and index
38:4~~7~~6	- Empty
47	- Debug Address
48:52	- 5 debug export (interpret as normal memory export)
53:59	- Empty
60	- export addressing mode
61	- Empty
62	- position
63	- sprite size export that goes with position export (X= point size, Y= edge flag is bit 0, Z= VtxKill is bitwise OR of bits 30:0. Any bit other than sign means VtxKill.)

Formatted: Bullets and Numbering

~~18.2~~17.2 Pixel Shading

0	- Color for buffer 0 (primary)
1	- Color for buffer 1
2	- Color for buffer 2
3	- Color for buffer 3
4:15	- Empty
16	- Buffer 0 Color/Fog (primary)
17	- Buffer 1 Color/Fog

18	- Buffer 2 Color/Fog
19	- Buffer 3 Color/Fog
20:31	- Empty
32	- Export Address
33:37	- 5 exports for multipass pixel shaders.
38:4~~7~~6	- Empty
47	- Debug Address
48:52	- 5 debug exports (interpret as normal memory export)
60	- export addressing mode
61	- Z for primary buffer (Z exported to 'alpha' component)
62:63	- Empty

Formatted: Bullets and Numbering

~~19.~~18. Special Interpolation modes

~~19.1~~18.1 Real time commands

We are unable to use the parameter memory since there is no way for a command stream to write into it. Instead we need to add three ~~16x128~~ 4x128 memories (one for each of three vertices x ~~16~~ 4 interpolants). These will be mapped onto the register bus and written by type 0 packets, and output to the the parameter busses (the sequencer and/or PA need to be able to address the realtime parameter memory as well as the regular parameter store~~. For higher performance we should be able able to view them as two banks of 16 and do double buffering allowing one to be loaded, while the other is rasterized with. Most overlay shaders will need 2 or 4 scalar coordinates, one option might be to restrict the memory to 16x64 or 32x64 allowing only two interpolated scalars per cycle, the only problem I see with this is, if we view support for 16 vector-4 interpolants important (true only if we map Microsoft's high priority stream to the realtime stream), then the PA/sequencer need to support a realtime-specific mode where we need to address 32 vectors of parameters instead of 16~~. This mode is triggered by the primitive type: REAL TIME. The actual memories are in the in the SX blocks. The parameter data memories are hooked on the RBBM bus and are loaded by the CP using register mapped memory.

Formatted: Bullets and Numbering

~~19.2~~18.2 Sprites/ XY screen coordinates/ FB information

XY screen coordinates may be needed in the shader program. This functionality is controlled by the param_gen~~n_I0~~ register (in SQ) in conjunction with the SND_XY register (in SC) and the param_gen_pos. Also it is possible to send the faceness information (for OGL front/back special operations) to the shader using the same control register. Here is a list of all the modes and how they interact together:

The Data is going to be written in the register specified by the param_gen_pos register.

Param_Gen_~~I0~~ disable, snd_xy disable = No modification
Param_Gen_~~I0~~ disable, snd_xy enable = No modification
Param_Gen_~~I0~~ enable, snd_xy disable = Sign(faceness)garbage,(Sign Point)garbage,Sign(Line)s, t
Param_Gen_~~I0~~ enable, snd_xy enable = Sign(faceness)screenX,(Sign Point)screenY,Sign(Line)s, t

In other words,

The generated vector is (X in RED, Y in GREEN, S in BLUE and T in ALPHA):
X,Y,S,T
These values are always supposed to be positive and any shader use of them should use the ABS function (as their sign bits will now be used for flags).
SignX = BackFacing
SignY = Point Primitive
SignS = Line Primitive
SignT = currently unused as a flag.

If !Point & !Line, then it is a Poly.

I would assume that one implementation which allows for generic texture lookup (using 3D maps) for poly stipple and AA for the driver would be

```
if(Y<0) {
        R = 0.0 (Point)
        } else if (S < 0) {
                R = 1.0 (Line)
        } else {
                R = 2.0 (Poly)
}
```

~~19.3~~18.3 Auto generated counters

Formatted: Bullets and Numbering

In the cases we are dealing with multipass shaders, the sequencer is going to generate a vector count to be able to both use this count to write the 1st pass data to memory and then use the count to retrieve the data on the 2nd pass. The count is always generated in the same way but it is passed to the shader in a slightly different way depending on the shader type (pixel or vertex). This is toggled on and off using the GEN_INDEX_PIX/VTX register. The sequencer is going to keep two counters, one for pixels and one for vertices. Every time a full vector of vertices or pixels is written to the GPRs the counter is incremented. Every time a RST_PIX_COUNT or RST_VTX_COUNT events are received, the corresponding counter is reset. While there is only one count broadcast to the GPRs, the LSB are hardwired to specific values making the index different for all elements in the vector. Since the count must be different for all pixels/vertices and the 4 LSBs (16 positions) are hardwired to the corresponding shader unit the SQ has two choices:

1) Maintain a 19 bit counter that counts the vectors of 64. In this case the phase must be appended to the count before the count is broadcast to the SPs:

Counter (19 bits)	Phase (2 bits)	Hardwired (4 bits)

2) Maintain a 21 bits counter that counts sub-vectors of 16. In this case only the counter is sent to the Sps:

Counter (21 bits)	Hardwired (4 bits)

~~19.3.1~~18.3.1 *Vertex shaders*

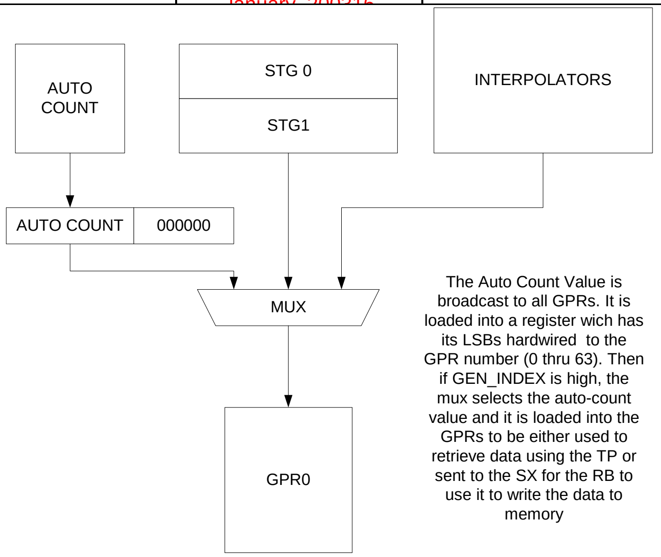
Formatted: Bullets and Numbering

In the case of vertex shaders, if GEN_INDEX_VTX is set, the data will be put into the x field of the third register (it means that the compiler must allocate 3 GPRs in all multipass vertex shader modes).

~~19.3.2~~18.3.2 *Pixel shaders*

Formatted: Bullets and Numbering

In the case of pixel shaders, if GEN_INDEX_PIX is set, the data will be put in the x field of the param_gen_pos+1 register.



The Auto Count Value is broadcast to all GPRs. It is loaded into a register wich has its LSBs hardwired to the GPR number (0 thru 63). Then if GEN_INDEX is high, the mux selects the auto-count value and it is loaded into the GPRs to be either used to retrieve data using the TP or sent to the SX for the RB to use it to write the data to memory

Figure 12: GPR input mux Control

Formatted: Bullets and Numbering

~~20.~~19. State management

Every clock, the sequencer will report to the CP the oldest states still in the pipe. These are the states of the programs as they enter the last ALU clause.

Formatted: Bullets and Numbering

~~20.1~~19.1 Parameter cache synchronization

In order for the sequencer not to begin a group of pixels before the associated group of vertices has finished, the sequencer will keep a 6 bit count per state (for a total of 8 counters). These counters are initialized to 0 and every time a vertex shader exports its data TO THE PARAMETER CACHE, the corresponding pointer is incremented. When the SC sends a new vector of pixels with the SC_SQ_new_vector bit asserted, the sequencer will first check if the count is greater than 0 before accepting the transmission (it will in fact accept the transmission but then lower its ready to receive). Then the sequencer waits for the count to go to one and decrements it. The sequencer can then issue the group of pixels to the interpolators. Every time the state changes, the new state counter is initialized to 0.

Formatted: Bullets and Numbering

~~21.~~20. XY Address imports

The SC will be able to send the XY addresses to the GPRs. It does so by interleaving the writes of the IJs (to the IJ buffer) with XY writes (to the XY buffer). Then when writing the data to the GPRs, the sequencer is going to interpolate the IJ data or pass the XY data thru a Fix→float converter and expander and write the converted values to the GPRs. The Xys are currently SCREEN SPACE COORDINATES. The values in the XY buffers will wrap. See section 18.2~~18.219.2~~ for details on how to control the interpolation in this mode.

Formatted: Bullets and Numbering

~~21.1~~20.1 Vertex indexes imports

In order to import vertex indexes, we have 16 8x96 staging registers. These are loaded one line at a time by the VGT block (96 bits). They are loaded in floating point format and can be transferred in 4 or 8 clocks to the GPRs.

Formatted: Bullets and Numbering

~~22.~~21. Registers

Please see the auto-generated web pages for register definitions.

~~23.~~22. Interfaces

Formatted: Bullets and Numbering

~~23.1~~22.1 External Interfaces

Whenever an x is used, it means that the bus is broadcast to all units of the same name. For example, if a bus is named SQ→SPx it means that SQ is going to broadcast the same information to all SP instances.

~~23.2~~22.2 SC to SP Interfaces

Formatted: Bullets and Numbering

~~23.2.1~~22.2.1 *SC_SP#*

Formatted: Bullets and Numbering

There is one of these interfaces at front of each of the SP (buffer to stage pixel interpolators). This interface transmits the I,J data for pixel interpolation. For the entire system, two quads per clock are transferred to the 4 SPs, so each of these 4 interfaces transmits one half of a quad per clock. The interface below describes a half of a quad worth of data.
The actual data which is transferred per quad is
Ref Pix I => S4.20 Floating Point I value *4
Ref Pix J => S4.20 Floating Point J value *4

This equates to a total of 200 bits which transferred over 2 clocks
and therefor needs an interface 100 bits wide

Additionally, X,Y data (12-bit unsigned fixed) is conditionally sent across this data bus over the same wires in an additional clock. The X,Y data is sent on the lower 24 bits of the data bus with faceness in the msb.
Transfers across these interfaces are synchronized with the SC_SQ IJ Control Bus transfers.

The data transfer across each of these busses is controlled by a IJ_BUF_INUSE_COUNT in the SC. Each time the SC has sent a pixel vector's worth of data to the SPs, he will increment the IJ_BUF_INUSE_COUNT count. Prior to sending the next pixel vectors data, he will check to make sure the count is less than MAX_BUFER_MINUS_2, if not the SC will stall until the SQ returns a pipelined pulse to decrement the count when he has scheduled a buffer free. Note: We could/may optimize for the case of only sending only IJ to use all the buffers to pre-load more. Currently it is planned for the SP to hold 2 double buffers of I,J data and two buffers of X,Y data, so if either X,Y or Centers and Centroids are on, then the SC can send two Buffers.

In at least the initial version, the SC shall send 16 quads per pixel vector even if the vector is not full. This will increment buffer write address pointers correctly all the time. (We may revisit this for both the SX,SP,SQ and add a EndOfVector signal on all interfaces to quit early. We opted for the simple mode first with a belief that only the end of packet and multiple new vector signals should cause a partial vector and that this would not really be significant performance hit.)

Name	Bits	Description
SC_SP#_data	100	IJ information sent over 2 clocks (or X,Y in 24 LSBs with faceness in upper bit) **Type 0 or 1**, First clock I, second clk J Field ULC URC LLC LRC Bits [63:39] [38:26] [25:13] [12:0] Format SE4M20 SE4M20 SE4M20 SE4M20 **Type 2** Field Face X Y Bits [24] [23:12] [11:0] Format Bit Unsigned Unsigned
SC_SP#_valid	1	Valid
SC_SP#_last_quad_data	1	This bit will be set on the last transfer of data per quad.
SC_SP#_type	2	0 -> Indicates centroids 1 -> Indicates centers 2 -> Indicates X,Y Data and faceness on data bus The SC shall look at state data to determine how many types to send for the

		interpolation process.

The # is included for clarity in the spec and will be replaced with a prefix of u#_ in the verilog module statement for the SC and the SP block will have neither because the instantiation will insert the prefix.

Formatted: Bullets and Numbering

~~23.2.2~~22.2.2 SC_SQ

This is the control information sent to the sequencer in order to synchronize and control the interpolation and/or loading data into the GPRs needed to execute a shader program on the sent pixels. This data will be sent over two clocks per transfer with 1 to 16 transfers. Therefore the bus (approx 108 bits) could be folded in half to approx 54 bits.

Name	Bits	Description
SC_SQ_data	46	Control Data sent to the SQ 1 clk transfers Event – valid data consist of event_id and state_id. Instruct SQ to post an event vector to send state id and event_id through request fifo and onto the reservation stations making sure state id and/or event_id gets back to the CP. Events only follow end of packets so no pixel vectors will be in progress. Empty Quad Mask – Transfer Control data consisting of pc_dealloc or new_vector. Receipt of this is to transfer pc_dealloc or new_vector without any valid quad data. New vector will always be posted to request fifo and pc_dealloc will be attached to any pixel vector outstanding or posted in request fifo if no valid quad outstanding. 2 clk transfers Quad Data Valid – Sending quad data with or without new_vector or pc_dealloc. New vector will be posted to request fifo with or without a pixel vector and pc_dealloc will be posted with a pixel vector unless none is in progress. In this case the pc_dealloc will be posted in the request queue. Filler quads will be transferred with The Quad mask set but the pixel corresponding pixel mask set to zero.
SC_SQ_valid	1	SC sending valid data, 2nd clk could be all zeroes

SC_SQ_data – first clock and second clock transfers are shown in the table below.

Name	BitField	Bits	Description
1st Clock Transfer			
SC_SQ_event	0	1	This transfer is a 1 clock event vector Force quad_mask = new_vector=pc_dealloc=0
SC_SQ_event_id	[5:1]	4	This field identifies the event 0 => denotes an End Of State Event 1

			=> TBD
SC_SQ_state_id	[8:6]	3	State/constant pointer (6*3+3)
SC_SQ_pc_dealloc	[11:9]	3	Deallocation token for the Parameter Cache
SC_SQ_new_vector	12	1	The SQ must wait for Vertex shader done count > 0 and after dispatching the Pixel Vector the SQ will decrement the count.
SC_SQ_quad_mask	[16:13]	4	Quad Write mask left to right SP0 => SP3
SC_SQ_end_of_prim	17	1	End Of the primitive
SC_SQ_pix_mask	[33:18]	16	Valid bits for all pixels SP0=>SP3 (UL,UR,LL,LR)
SC_SQ_provok_vtx	[35:34]	2	Provoking vertex for flat shading
SC_SQ_lod_correct_0	[44:36]	9	LOD correction for quad 0 (SP0) (9 bits per quad)
SC_SQ_lod_correct_1	[53:45]	9	LOD correction for quad 1 (SP1) (9 bits per quad)
2nd Clock Transfer			
SC_SQ_lod_correct_2	[8:0]	9	LOD correction for quad 2 (SP2) (9 bits per quad)
SC_SQ_lod_correct_3	[17:9]	9	LOD correction for quad 3 (SP3) (9 bits per quad)
SC_SQ_pc_ptr0	[28:18]	11	Parameter Cache pointer for vertex 0
SC_SQ_pc_ptr1	[39:29]	11	Parameter Cache pointer for vertex 1
SC_SQ_pc_ptr2	[50:40]	11	Parameter Cache pointer for vertex 2
SC_SQ_prim_type	[53:51]	3	Stippled line and Real time command need to load tex cords from alternate buffer 000: Sprite (point) 001: Line 010: Tri_rect 100: Realtime Sprite (point) 101: Realtime Line 110: Realtime Tri_rect

Name	Bits	Description
SQ_SC_free_buff	1	Pipelined bit that instructs SC to decrement count of buffers in use.
SQ_SC_dec_cntr_cnt	1	Pipelined bit that instructs SC to decrement count of new vector and/or event sent to prevent SC from overflowing SQ interpolator/Reservation request fifo.

The scan converter will submit a partial vector whenever:

1.) He gets a primitive marked with an end of packet signal.
2.) A current pixel vector is being assembled with at least one or more valid quads and the vector has been marked for deallocate when a primitive marked new_vector arrives. The Scan Converter will submit a partial vector (up to 16quads with zero pixel mask to fill out the vector) prior to submitting the new_vector marker\primitive.

(This will prevent a hang which can be demonstrated when all primitives in a packet three vectors are culled except for a one quad primitive that gets marked pc_dealloc (vertices maximum size). In this case two new_vectors are submitted and processed, but then one valid quad with the pc_dealloc creates a vector and then the new would wait for another vertex vector to be processed, but the one being waited for could never export until the pc_dealloc signal made it through and thus the hang.)

~~23.2.3~~22.2.3 SQ to SX(SP): Interpolator bus

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SPx_interp_flat_vtx	SQ→SPx	2	Provoking vertex for flat shading
SQ_SPx_interp_flat_gouraud	SQ→SPx	1	Flat or gouraud shading
SQ_SPx_interp_cyl_wrap	SQ→SPx	4	Wich channel needs to be cylindrical wrapped
SQ_SPx_interp_param_gen	SQ→SPx	1	Generate Parameter
SQ_SPx_interp_prim_type	SQ→SPx	2	Bits [1:0] of primitive type sent by SC
SQ_SPx_interp_buff_swap	SQ→SPx	1	Swapp IJ buffers
SQ_SPx_interp_IJ_line	SQ→SPx	2	IJ line number
SQ_SPx_interp_mode	SQ→SPx	1	Center/Centroid sampling
SQ_SXx_pc_ptr0	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_pc_ptr1	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_pc_ptr2	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_rt_sel	SQ→SXx	1	Selects between RT and Normal data (Bit 2 of prim type)
SQ_SX0_pc_wr_en	SQ→SX0	8	Write enable for the PC memories
SQ_SX1_pc_wr_en	SQ→SX1	8	Write enable for the PC memories
SQ_SXx_pc_wr_addr	SQ→SXx	7	Write address for the PCs
SQ_SXx_pc_channel_mask	SQ→SXx	4	Channel mask
SQ_SXx_pc_ptr_valid	SQ→SXx	1	Read pointers are valid.
SQ_SPx_interp_valid	SQ→SPx	1	Interpolation control valid

~~23.2.4~~22.2.4 SQ to SP: Staging Register Data

Formatted: Bullets and Numbering

This is a broadcast bus that sends the VSISR information to the staging registers of the shader pipes.

Name	Direction	Bits	Description
SQ_SPx_vsr_data	SQ→SPx	96	Pointers of indexes or HOS surface information
SQ_SPx_vsr_wrt_addr	SQ→SPx	3	Staging register write address
SQ_SPx_vsr_rd_addr~~SQ_SPx_vsr_double~~	SQ→SPx	3~~1~~	Staging register read address~~0: Normal 96 bits per vert 1: double 192 bits per vert~~
SQ_SP0_ vsr_valid	SQ→SP0	1	Data is valid
SQ_SP1_ vsr_ valid	SQ→SP1	1	Data is valid
SQ_SP2_ vsr_ valid	SQ→SP2	1	Data is valid
SQ_SP3_ vsr_ valid	SQ→SP3	1	Data is valid
SQ_SPx_vsr_read	SQ→SPx	1	Increment the read pointers

~~23.2.5~~22.2.5 VGT to SQ : Vertex interface

Formatted: Bullets and Numbering

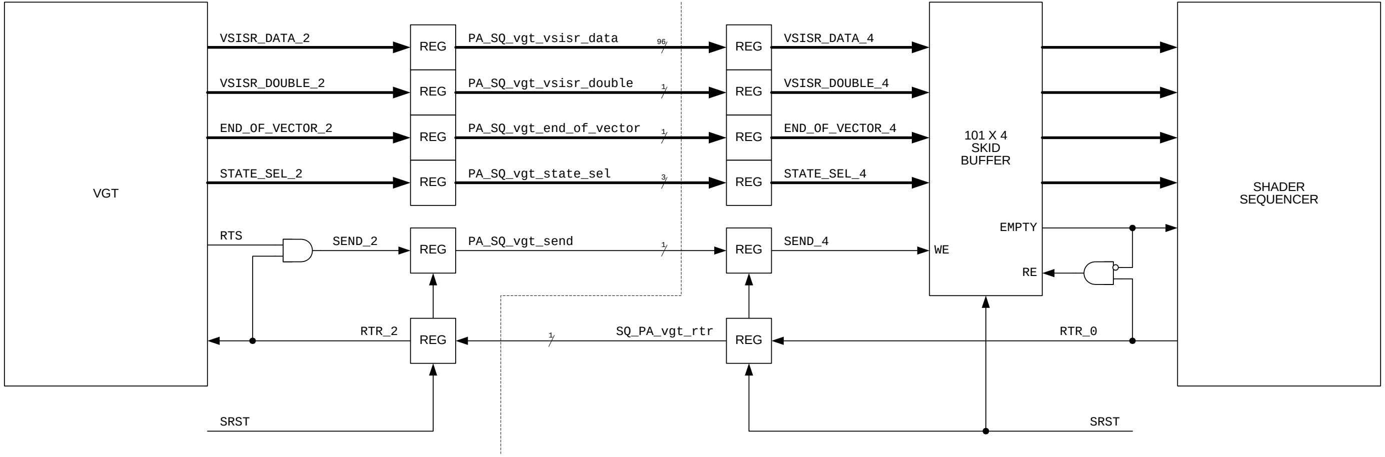
~~23.2.5.1~~22.2.5.1 Interface Signal Table

The area difference between the two methods is not sufficient to warrant complicating the interface or the state requirements of the VSISRs. **Therefore, the POR for this interface is that the VGT will transmit the data to the VSISRs (via the Shader Sequencer) in full, 32-bit floating-point format.** The VGT can transmit up to six 32-bit floating-point values to each VSISR where four or more values require two transmission clocks. The data bus is 96 bits wide. In the case where an event is sent the 5 LSBs of VGT_SQ_vsisr_data contain the eventID.

Name	Bits	Description
VGT_SQ_vsisr_data	96	Pointers of indexes or HOS surface information
VGT_SQ_event	1	VGT is sending an event
VGT_SQ_vsisr_continued	1	0: Normal 96 bits per vert 1: double 192 bits per vert
VGT_SQ_end_of_vtx_vect	1	Indicates the last VSISR data set for the current process vector (for double vector data, "end_of_vector" is set on the first vector)
VGT_SQ_indx_valid	1	Vsisr data is valid
VGT_SQ_state	3	Render State (6*3+3 for constants). This signal is guaranteed to be correct when “VGT_SQ_vgt_end_of_vector” is high.
VGT_SQ_send	1	Data on the VGT_SQ is valid receive (see write-up for standard R400 SEND/RTR interface handshaking)
SQ_VGT_rtr	1	Ready to receive (see write-up for standard R400 SEND/RTR interface handshaking)

~~23.2.5.2~~22.2.5.2 Interface Diagrams

Formatted: Bullets and Numbering



VGT
VSISR_DATA_2
VSISR_DOUBLE_2
END_OF_VECTOR_2
STATE_SEL_2
RTS
SEND_2
RTR_2
SRST
REG
PA_SQ_vgt_vsisr_data
96
PA_SQ_vgt_vsisr_double
1
PA_SQ_vgt_end_of_vector
1
PA_SQ_vgt_state_sel
3
PA_SQ_vgt_send
1
SQ_PA_vgt_rtr
1
VSISR_DATA_4
VSISR_DOUBLE_4
END_OF_VECTOR_4
STATE_SEL_4
SEND_4
101 X 4
SKID
BUFFER
WE
EMPTY
RE
RTR_0
SRST
SHADER
SEQUENCER

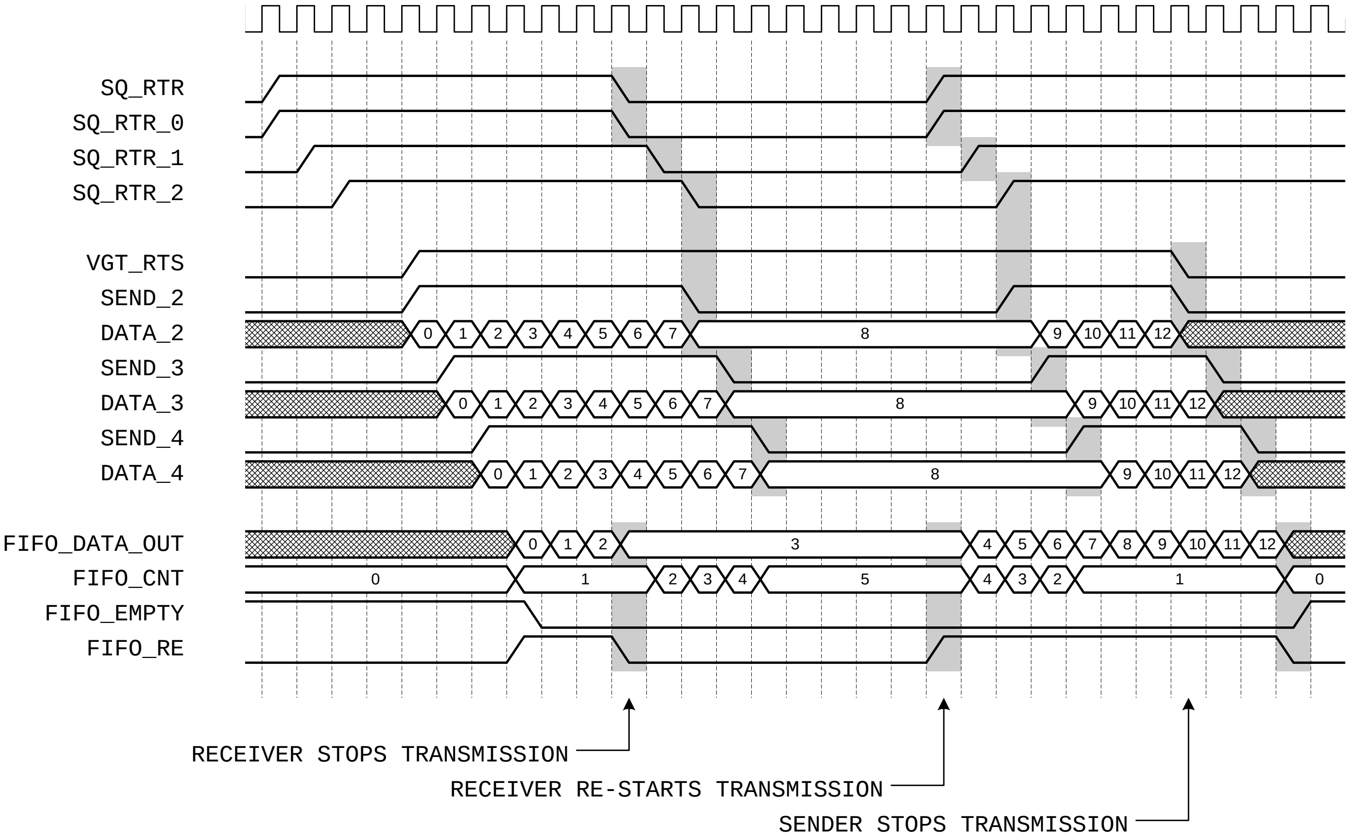


Figure 1. Detailed Logical Diagram for PA_SQ_vgt Interface.

~~23.2.6~~22.2.6 *SQ to SX: Control bus*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SXx_exp_type	SQ→SXx	2	00: Pixel without z (1 to 4 buffers) 01: Pixel with z (1 to 4 buffers) 10: Position (1 or 2 results) 11: Pass thru (4,8 or 12 results aligned)
SQ_SXx_exp_number	SQ→SXx	2	Number of locations needed in the export buffer (encoding depends on the type see bellow).
SQ_SXx_exp_alu_id	SQ→SXx	1	ALU ID
SQ_SXx_exp_valid	SQ→SXx	1	Valid bit
SQ_SXx_exp_state	SQ→SXx	3	State Context
SQ_SXx_free_done	SQ→SXx	1	Pulse that indicates that the previous export is finished **from the point of view of the SP. This does not necessarily mean that the data has been transferred to RB or PA, or that the space in export buffer for that particular vector thread has been freed up.**
SQ_SXx_free_alu_id	SQ→SXx	1	ALU ID

Depending on the type the number of export location changes:

- Type 00 : Pixels without Z
  - 00 = 1 buffer
  - 01 = 2 buffers
  - 10 = 3 buffers
  - 11 = 4 buffer
- Type 01: Pixels with Z
  - 00 = 2 Buffers (color + Z)
  - 01 = 3 buffers (2 color + Z)
  - 10 = 4 buffers (3 color + Z)
  - 11 = 5 buffers (4 color + Z)
- Type 10 : Position export
  - 00 = 1 position
  - 01 = 2 positions
  - 1X = Undefined
- Type 11: Pass Thru
  - 00 = 4 buffers
  - 01 = 8 buffers
  - 10 = 12 buffers
  - 11 = Undefined

Below the thick black line is the end of transfer packet that tells the SX that a given export is finished. The report packet **will always arrive either before or at the same time than the next export to the same ALU id**.

~~23.2.7~~22.2.7 *SX to SQ : Output file control*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SXx_SQ_exp_count_rdy	SXx→SQ	1	Raised by SX0 to indicate that the following two fields reflect the result of the most recent export
SXx_SQ_exp_pos_avail	SXx→SQ	2	Specifies whether there is room for another position. 00 : 0 buffers ready 01 : 1 buffer ready 10 : 2 or more buffers ready
SXx_SQ_exp_buf_avail	SXx→SQ	7	Specifies the space available in the output buffers. 0: buffers are full 1: 2K-bits available (32-bits for each of the 64

			pixels in a clause) ... 64: 128K-bits available (16 128-bit entries for each of 64 pixels) 65-127: RESERVED

~~23.2.8~~22.2.8 *SQ to TP: Control bus*

Formatted: Bullets and Numbering

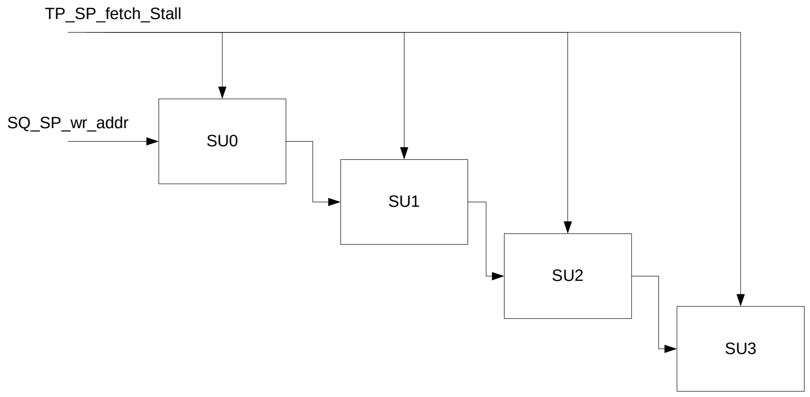
Once every clock, the fetch unit sends to the sequencer on which RS line it is now working and if the data in the GPRs is ready or not. This way the sequencer can update the fetch valid bits flags for the reservation station. The sequencer also provides the instruction and constants for the fetch to execute and the address in the register file where to write the fetch return data.

Name	Direction	Bits	Description
TPx_SQ_data_rdy	TPx→ SQ	1	Data ready
TPx_SQ_rs_line_num	TPx→ SQ	6	Line number in the Reservation station
TPx_SQ_type	TPx→ SQ	1	Type of data sent (0:PIXEL, 1:VERTEX)
SQ_TPx_send	SQ→TPx	1	Sending valid data
SQ_TPx_const	SQ→TPx	48	Fetch state sent over 4 clocks (192 bits total)
SQ_TPx_instr	SQ→TPx	24	Fetch instruction sent over 4 clocks
SQ_TPx_end_of_group	SQ→TPx	1	Last instruction of the group
SQ_TPx_Type	SQ→TPx	1	Type of data sent (0:PIXEL, 1:VERTEX)
SQ_TPx_gpr_phase	SQ→TPx	2	Write phase signal
SQ_TP0_lod_correct	SQ→TP0	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP0_pix_mask	SQ→TP0	4	Pixel mask 1 bit per pixel
SQ_TP1_lod_correct	SQ→TP1	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP1_pix_mask	SQ→TP1	4	Pixel mask 1 bit per pixel
SQ_TP2_lod_correct	SQ→TP2	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP2_pix_mask	SQ→TP2	4	Pixel mask 1 bit per pixel
SQ_TP3_lod_correct	SQ→TP3	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP3_pix_mask	SQ→TP3	4	Pixel mask 1 bit per pixel
SQ_TPx_rs_line_num	SQ→TPx	6	Line number in the Reservation station
SQ_TPx_write_gpr_index	SQ->TPx	7	Index into Register file for write of returned Fetch Data
SQ_TPx_ctx_id	SQ→TPx	3	The state context ID (needed for multisample resolves)

~~23.2.9~~22.2.9 *TP to SQ: Texture stall*

Formatted: Bullets and Numbering

The TP sends this signal to the SQ and the SPs when its input buffer is full.



Name	Direction	Bits	Description
TP_SQ_fetch_stall	TP→ SQ	1	Do not send more texture request if asserted

~~23.2.10 SQ to SP: Texture stall~~

Formatted: Bullets and Numbering

~~23.2.11~~22.2.10 SQ to SP: GPR and auto counter

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SPx_gpr_wr_addr	SQ→SPx	7	Write address
SQ_SPx_gpr_rd_addr	SQ→SPx	7	Read address
SQ_SPx_gpr_rd_en	SQ→SPx	1	Read Enable
SQ_SP0_gpr_pspv_wr_en	SQ→SPx	4	Write Enable for the GPRs of SP0 for PS and PV
SQ_SP1_gpr_pspv_wr_en	SQ→SPx	4	Write Enable for the GPRs of SP1 for PS and PV
SQ_SP2_gpr_pspv_wr_en	SQ→SPx	4	Write Enable for the GPRs of SP2 for PS and PV
SQ_SP3_gpr_pspv_wr_en	SQ→SPx	4	Write Enable for the GPRs of SP3 for PS and PV
SQ_SP0_gpr_int_wr_en	SQ→SPx	1	Write Enable for the GPRs of SP0 for Inputs (interp/vtx)
SQ_SP1_gpr_int_wr_en	SQ→SPx	1	Write Enable for the GPRs of SP1 for Inputs (interp/vtx)
SQ_SP2_gpr_int_wr_en	SQ→SPx	1	Write Enable for the GPRs of SP2 for Inputs (interp/vtx)
SQ_SP3_gpr_int_wr_en~~SQ_SP3_gpr_wr_en~~	SQ→SPx~~SQ→SPx~~	1~~4~~	Write Enable for the GPRs of SP3 for Inputs (interp/vtx)~~Write Enable for the GPRs of SP3~~
SQ_SPx_gpr_phase	SQ→SPx	2	The phase mux (arbitrates between inputs, ALU SRC reads and writes)
SQ_SPx_channel_mask	SQ→SPx	4	The channel mask
SQ_SPx_gpr_input_sel	SQ→SPx	2	When the phase mux selects the inputs this tells from which source to read from: Interpolated data, VTX0, VTX1, autogen counter.
SQ_SPx_auto_count	SQ→SPx	21	Auto count generated by the SQ, common for all shader pipes

~~23.2.12~~22.2.11 SQ to SPx: Instructions

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SPx_instr_start	SQ→SPx	1	Instruction start
SQ_SP_instr	SQ→SPx	24	Transferred over 4 cycles 0: SRC A Negate Argument Modifier 0:0 SRC A Abs Argument Modifier 1:1 SRC A Swizzle 9:2 Vector Dst 15:10 Per channel Select 23:16 00: GPR 01: PV 10: PS 11: Constant (if 11 has to be 11 for all channels) --- 1: SRC B Negate Argument Modifier 0:0 SRC B Abs Argument Modifier 1:1 SRC B Swizzle 9:2 Scalar Dst 15:10 Per channel Select 23:16 00: GPR 01: PV 10: PS 11: Constant (if 11 has to be 11 for all channels) --- 2: SRC C Negate Argument Modifier 0:0 SRC C Abs Argument Modifier 1:1 SRC C Swizzle 9:2 Unused 15:10 Per channel Select 23:16 00: GPR 01: PV 10: PS 11: Constant (if 11 has to be 11 for all channels) --- 3: Vector Opcode 4:0 Scalar Opcode 10:5 Vector Clamp 11:11 Scalar Clamp 12:12 Vector Write Mask 16:13 Scalar Write Mask 20:17 Unused 23:21
SQ_SP0_pred_override	SQ→SP0	4	0: Use per channel RGBA field (enables the per channel logic, if not set only pay attention to the 11 seting). 1: Use GPR
SQ_SP1_pred_override	SQ→SP1	4	0: Use per channel RGBA field (enables the per channel logic, if not set only pay attention to the 11 seting). 1: Use GPR
SQ_SP2_pred_override	SQ→SP2	4	0: Use per channel RGBA field (enables the per channel logic, if not set only pay attention to the 11 seting). 1: Use GPR
SQ_SP3_pred_override	SQ→SP3	4	0: Use per channel RGBA field (enables the per channel logic, if not set only pay attention to the 11

			seting). 1: Use GPR
SQ_SPx_exp_id	SQ→SPx	1	GPR ID
SQ_SPx_exporting	SQ→SPx	1	0: Not Exporting 1: Exporting
SQ_SPx_stall	SQ→SPx	1	Stall signal
SQ_SPx_Waterfall	SQ→SPx	2	Use the incoming constant instead of the registered one for the next group of 16. 0 : Normal mode 1: Waterfall on SRCA 2: Waterfall on SRCB 3: Waterfall on SRCC

~~23.2.13~~22.2.12 *SQ to SX: write mask interface (must be aligned with the SP data)*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SX0_write_mask	SQ→SP0	8	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock. This is for the data coming of SP0 and SP2.
SQ_SX1_ write_mask	SQ→SP1	8	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock. This is for the data coming of SP1 and SP3.

~~23.2.14~~22.2.13 *SP to SQ: Constant address load/ Predicate Set/Kill set*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SP0_SQ_const_addr	SP0→SQ	36	Constant address load / predicate vector load (4 bits only)/ Kill vector load (4 bits only) to the sequencer
SP0_SQ_valid	SP0→SQ	1	Data valid
SP1_SQ_const_addr	SP1→SQ	36	Constant address load / predicate vector load (4 bits only)/ Kill vector load (4 bits only) to the sequencer
SP1_SQ_valid	SP1→SQ	1	Data valid
SP2_SQ_const_addr	SP2→SQ	36	Constant address load / predicate vector load (4 bits only)/ Kill vector load (4 bits only) to the sequencer
SP2_SQ_valid	SP2→SQ	1	Data valid
SP3_SQ_const_addr	SP3→SQ	36	Constant address load / predicate vector load (4 bits only)/ Kill vector load (4 bits only) to the sequencer
SP3_SQ_valid	SP3→SQ	1	Data valid
SP0_SQ_data_type	SP→SQ	2	Data Type 0: Constant Load 1: Predicate Set 2: Kill vector load

Because of the sharing of the bus none of the MOVA, PREDSET or KILL instructions may be coissued.

~~23.2.15~~22.2.14 *SQ to SPx: constant broadcast*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_SPx_const	SQ→SPx	128	Constant broadcast

~~23.2.16~~22.2.15 *SQ to CP: RBBM bus*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_RBB_rs	SQ→CP	1	Read Strobe
SQ_RBB_rd	SQ→CP	32	Read Data
SQ_RBBM_nrtrtr	SQ→CP	1	Optional
SQ_RBBM_rtr	SQ→CP	1	Real-Time (Optional)

~~23.2.17~~22.2.16 *CP to SQ: RBBM bus*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
rbbm_we	CP→SQ	1	Write Enable
rbbm_a	CP→SQ	15	Address -- Upper Extent is TBD (16:2)
rbbm_wd	CP→SQ	32	Data
rbbm_be	CP→SQ	4	Byte Enables
rbbm_re	CP→SQ	1	Read Enable
rbb_rs0	CP→SQ	1	Read Return Strobe 0
rbb_rs1	CP→SQ	1	Read Return Strobe 1
rbb_rd0	CP→SQ	32	Read Data 0
rbb_rd1	CP→SQ	32	Read Data 0
RBBM_SQ_soft_reset	CP→SQ	1	Soft Reset

~~23.2.18~~22.2.17 *SQ to CP: State report*

Formatted: Bullets and Numbering

Name	Direction	Bits	Description
SQ_CP_vs_event	SQ→CP	1	Vertex Shader Event
SQ_CP_vs_eventid	SQ→CP	5	Vertex Shader Event ID
SQ_CP_ps_event	SQ→CP	1	Pixel Shader Event
SQ_CP_ps_eventid	SQ→CP	5	Pixel Shader Event ID

~~23.3~~22.3 Example of control flow program execution

Formatted: Bullets and Numbering

We now provide some examples of execution to better illustrate the new design.

Given the program:

Alu 0
Alu 1
Tex 0
Tex 1
Alu 3 Serial
Alu 4
Tex 2
Alu 5
Alu 6 Serial
Tex 3
Alu 7
Alloc Position 1 buffer
Alu 8 Export
Tex 4
Alloc Parameter 3 buffers
Alu 9 Export 0
Tex 5
Alu 10 Serial Export 2
Alu 11 Export 1 End

Would be converted into the following CF instructions:

```
Execute 0 Alu 0 Alu 0 Tex 0 Tex 1 Alu 0 Alu 0 Tex 0 Alu 1 Alu 0 Tex
Execute 0 Alu
Alloc Position 1
Execute 0 Alu 0 Tex
Alloc Param 3
Execute_end 0 Alu 0 Tex 1 Alu 0 Alu
```

And the execution of this program would look like this:

Put thread in Vertex RS:

Control Flow Instruction Pointer (12 bits), (CFP)
Execution Count Marker (3 or 4 bits), (ECM)
Loop Iterators (4x9 bits), (LI)
Call return pointers (4x12 bits), (CRP)
Predicate Bits(4x64 bits), (PB)
Export ID (1 bit), (EXID)
GPR Base Ptr (8 bits), (GPR)
Export Base Ptr (7 bits), (EB)
Context Ptr (3 bits).(CPTR)
LOD correction bits (16x6 bits) (LOD)

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	0	0	0	0	0	0	0	0	0

Valid Thread (VALID)
Texture/ALU engine needed (TYPE)
Texture Reads are outstanding (PENDING)
Waiting on Texture Read to Complete (SERIAL)
Allocation Wait (2 bits) (ALLOC)
  00 – No allocation needed
  01 – Position export allocation needed (ordered export)
  10 – Parameter or pixel export needed (ordered export)
  11 – pass thru (out of order export)
Allocation Size (4 bits) (SIZE)
Position Allocated (POS_ALLOC)
First thread of a new context (FIRST)
Last (1 bit), (LAST)

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	0	0	0	0	0	1	0

Then the thread is picked up for the execution of the first control flow instruction:

```
Execute 0 Alu 0 Alu 0 Tex 0 Tex 1 Alu 0 Alu 0 Tex 0 Alu 1 Alu 0 Tex
```

It executes the first two ALU instructions and goes back to the RS for a resource request change. Here is the state returned to the RS:

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	2	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	0	0	0	0	0	1	0

Then when the texture pipe frees up, the arbiter picks up the thread to issue the texture reads. The thread comes back in this state:

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	4	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	1	0	0	0	1	0

Because of the serial bit the arbiter must wait for the texture to return and clear the PENDING bit before it can pick the thread up. Lets say that the texture reads are complete, then the arbiter picks up the thread and returns it in this state:

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	6	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	0	0	0	0	0	1	0

Again the TP frees up, the arbiter picks up the thread and executes. It returns in this state:

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	7	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	0	0	0	1	0

Now, even if the texture has not returned we can still pick up the thread for ALU execution because the serial bit is not set. The thread will however come back to the RS for the second ALU instruction because it has the serial bit set.

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	8	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	1	0	0	0	1	0

As soon as the TP clears the pending bit the thread is picked up and returns:

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	9	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	0	0	0	0	0	1	0

Picked up by the TP and returns:
`Execute 0 Alu`

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
1	0	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	0	0	0	1	0

Picked up by the ALU and returns (lets say the TP has not returned yet):
`Alloc Position 1`

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
2	0	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	01	1	0	1	0

If the SX has the place for the export, the SQ is going to allocate and pick up the thread for execution. It returns to the RS in this state:

```
Execute 0 Alu 0 Tex
```

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
3	1	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	1	0	0	0	1	1	0

Now, since the TP has not returned yet, we must wait for it to return because we cannot issue multiple texture requests. The TP returns, clears the PENDING bit and we proceed:

```
Alloc Param 3
```

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
4	0	0	0	0	1	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	10	3	1	1	0

Once again the SQ makes sure the SX has enough room in the Parameter cache before it can pick up this thread.

```
Execute_end 0 Alu 0 Tex 1 Alu 0 Alu
```

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
5	1	0	0	0	1	0	100	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	1	0	0	0	1	1	0

This executes on the TP and then returns:

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
5	2	0	0	0	1	0	100	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	1	0	0	1	1	1

Waits for the TP to return because of the textures reads are pending (and SERIAL in this case). Then executes and does not return to the RS because the LAST bit is set. This is the end of this thread and before dropping it on the floor, the SQ notifies the SX of export completion.

Formatted: Bullets and Numbering

~~24.~~23. Open issues

Need to do some testing on the size of the register file as well as on the register file allocation method (dynamic VS static).

Saving power?

	ORIGINATE DATE 24 September, 2001	EDIT DATE 4 September, 2015	DOCUMENT-REV. NUM. GEN-CXXXXX-REVA	PAGE 1 of 56

Author: Laurent Lefebvre

Issue To:	**Copy No:**

R400 Sequencer Specification

SQ

Version 2.10

Overview: This is an architectural specification for the R400 Sequencer block (SEQ). It provides an overview of the required capabilities and expected uses of the block. It also describes the block interfaces, internal sub-blocks, and provides internal state diagrams.

AUTOMATICALLY UPDATED FIELDS:

Document Location:	C:\perforce\r400\doc_lib\design\blocks\sq\R400_Sequencer.doc
Current Intranet Search Title:	R400 Sequencer Specification

APPROVALS	
Name/Dept	Signature/Date

Remarks:

THIS DOCUMENT CONTAINS CONFIDENTIAL INFORMATION THAT COULD BE SUBSTANTIALLY DETRIMENTAL TO THE INTEREST OF ATI TECHNOLOGIES INC. THROUGH UNAUTHORIZED USE OR DISCLOSURE.

Table Of Contents

1. OVERVIEW 7
1.1 Top Level Block Diagram 9
1.2 Data Flow graph (SP) 10
1.3 Control Graph 11
2. INTERPOLATED DATA BUS 11
3. INSTRUCTION STORE 14
4. SEQUENCER INSTRUCTIONS 14
5. CONSTANT STORES 14
5.1 Memory organizations 14
5.2 Management of the Control Flow Constants 15
5.3 Management of the re-mapping tables 15
5.3.1 R400 Constant management 15
5.3.2 Proposal for R400LE constant management .... **Error! Bookmark not defined.**
5.3.3 Dirty bits 15
5.3.4 Free List Block 15
5.3.5 De-allocate Block 16
5.3.6 Operation of Incremental model 16
5.4 Constant Store Indexing 17
5.5 Real Time Commands 17
5.6 Constant Waterfalling 17
6. LOOPING AND BRANCHES 18
6.1 The controlling state. 18
6.2 The Control Flow Program 18
6.2.1 Control flow instructions table 19
6.3 Implementation 22
6.4 Data dependant predicate instructions 24
6.5 HW Detection of PV,PS 24
6.6 Register file indexing 24
6.7 Debugging the Shaders 25
6.7.1 Method 1: Debugging registers 25
6.7.2 Method 2: Exporting the values in the GPRs 25
7. PIXEL KILL MASK 25
8. MULTIPASS VERTEX SHADERS (HOS) .... ERROR! BOOKMARK NOT DEFINED.
9. REGISTER FILE ALLOCATION 25
10. FETCH ARBITRATION 27
11. ALU ARBITRATION 28
12. HANDLING STALLS 28
13. CONTENT OF THE RESERVATION STATION FIFOS 28
14. THE OUTPUT FILE 28
15. IJ FORMAT 28
15.1 Interpolation of constant attributes 29
16. STAGING REGISTERS 29

17. THE PARAMETER CACHE ... 30
17.1 Export restrictions ... 31
17.1.1 Pixel exports: ... 31
17.1.2 Vertex exports: ... 31
17.1.3 Pass thru exports: ... 31
17.2 Arbitration restrictions ... 32
18. EXPORT TYPES ... 32
18.1 Vertex Shading ... 32
18.2 Pixel Shading ... 33
19. SPECIAL INTERPOLATION MODES ... 33
19.1 Real time commands ... 33
19.2 Sprites/ XY screen coordinates/ FB information ... 33
19.3 Auto generated counters ... 34
19.3.1 Vertex shaders ... 34
19.3.2 Pixel shaders ... 34
20. STATE MANAGEMENT ... 35
20.1 Parameter cache synchronization ... 35
21. XY ADDRESS IMPORTS ... 35
21.1 Vertex indexes imports ... 35
22. REGISTERS ... 35
23. INTERFACES ... 36
23.1 External Interfaces ... 36
23.2 SC to SP Interfaces ... 36
23.2.1 SC_SP# ... 36
23.2.2 SC_SQ ... 37
23.2.3 SQ to SX(SP): Interpolator bus ... 39
23.2.4 SQ to SP: Staging Register Data ... 39
23.2.5 VGT to SQ : Vertex interface ... 39
23.2.6 SQ to SX: Control bus ... 43
23.2.7 SX to SQ : Output file control ... 43
23.2.8 SQ to TP: Control bus ... 44
23.2.9 TP to SQ: Texture stall ... 44
23.2.10 SQ to SP: Texture stall ... **Error! Bookmark not defined.**
23.2.11 SQ to SP: GPR and auto counter ... 45
23.2.12 SQ to SPx: Instructions ... 47
23.2.13 SQ to SX: write mask interface (must be aligned with the SP data) ... 50
23.2.14 SP to SQ: Constant address load/ Predicate Set/Kill set ... 51
23.2.15 SQ to SPx: constant broadcast ... 51
23.2.16 SQ to CP: RBBM bus ... 52
23.2.17 CP to SQ: RBBM bus ... 52
23.2.18 SQ to CP: State report ... 52

23.3 Example of control flow program execution..52
24. OPEN ISSUES..56

Revision Changes:

Rev 0.1 (Laurent Lefebvre) Date: May 7, 2001	First draft.
Rev 0.2 (Laurent Lefebvre) Date : July 9, 2001	Changed the interfaces to reflect the changes in the SP. Added some details in the arbitration section.
Rev 0.3 (Laurent Lefebvre) Date : August 6, 2001	Reviewed the Sequencer spec after the meeting on August 3, 2001.
Rev 0.4 (Laurent Lefebvre) Date : August 24, 2001	Added the dynamic allocation method for register file and an example (written in part by Vic) of the flow of pixels/vertices in the sequencer.
Rev 0.5 (Laurent Lefebvre) Date : September 7, 2001	Added timing diagrams (Vic)
Rev 0.6 (Laurent Lefebvre) Date : September 24, 2001	Changed the spec to reflect the new R400 architecture. Added interfaces.
Rev 0.7 (Laurent Lefebvre) Date : October 5, 2001	Added constant store management, instruction store management, control flow management and data dependant predication.
Rev 0.8 (Laurent Lefebvre) Date : October 8, 2001	Changed the control flow method to be more flexible. Also updated the external interfaces.
Rev 0.9 (Laurent Lefebvre) Date : October 17, 2001	Incorporated changes made in the 10/18/01 control flow meeting. Added a NOP instruction, removed the conditional_execute_or_jump. Added debug registers.
Rev 1.0 (Laurent Lefebvre) Date : October 19, 2001	Refined interfaces to RB. Added state registers.
Rev 1.1 (Laurent Lefebvre) Date : October 26, 2001	Added SEQ→SP0 interfaces. Changed delta precision. Changed VGT→SP0 interface. Debug Methods added.
Rev 1.2 (Laurent Lefebvre) Date : November 16, 2001	Interfaces greatly refined. Cleaned up the spec.
Rev 1.3 (Laurent Lefebvre) Date : November 26, 2001	Added the different interpolation modes.
Rev 1.4 (Laurent Lefebvre) Date : December 6, 2001	Added the auto incrementing counters. Changed the VGT→SQ interface. Added content on constant management. Updated GPRs.
Rev 1.5 (Laurent Lefebvre) Date : December 11, 2001	Removed from the spec all interfaces that weren't directly tied to the SQ. Added explanations on constant management. Added PA→SQ synchronization fields and explanation.
Rev 1.6 (Laurent Lefebvre) Date : January 7, 2002	Added more details on the staging register. Added detail about the parameter caches. Changed the call instruction to a Conditionnal_call instruction. Added details on constant management and updated the diagram.
Rev 1.7 (Laurent Lefebvre) Date : February 4, 2002	Added Real Time parameter control in the SX interface. Updated the control flow section.
Rev 1.8 (Laurent Lefebvre) Date : March 4, 2002	New interfaces to the SX block. Added the end of clause modifier, removed the end of clause instructions.
Rev 1.9 (Laurent Lefebvre) Date : March 18, 2002	Rearangement of the CF instruction bits in order to ensure byte alignement.
Rev 1.10 (Laurent Lefebvre) Date : March 25, 2002	Updated the interfaces and added a section on exporting rules.
Rev 1.11 (Laurent Lefebvre) Date : April 19, 2002	Added CP state report interface. Last version of the spec with the old control flow scheme
Rev 2.0 (Laurent Lefebvre) Date : April 19, 2002	New control flow scheme

Rev 2.01 (Laurent Lefebvre) Date : May 2, 2002	Changed slightly the control flow instructions to allow force jumps and calls.
Rev 2.02 (Laurent Lefebvre) Date : May 13, 2002	Updated the Opcodes. Added type field to the constant/pred interface. Added Last field to the SQ→SP instruction load interface.
Rev 2.03 (Laurent Lefebvre) Date : July 15, 2002	SP interface updated to include predication optimizations. Added the predicate no stall instructions,
Rev 2.04 (Laurent Lefebvre) Date :August 2, 2002	Documented the new parameter generation scheme for XY coordinates points and lines STs.
Rev 2.05 (Laurent Lefebvre) Date : September 10, 2002	Some interface changes and an architectural change to the auto-counter scheme.
Rev 2.06 (Laurent Lefebvre) Date : October 11, 2002	Widened the event interface to 5 bits. Some other little typos corrected.
Rev 2.07 (Laurent Lefebvre) Date : October 14, 2002	Loops, jumps and calls are now using a 13 bit address which allows to jump and call and loop around any control flow addresses (does not requires to be even anymore).
Rev 2.08 (Laurent Lefebvre) Date : October 16, 2002	Clarification updates after discussion with Clay.
Rev 2.09 (Laurent Lefebvre) Date : January 7, 2003	Corrected the SQ→SP staging register interface.
Rev 2.10 (Laurent Lefebvre) Date : April 8, 2003	Adding R500 modifications

1. Overview

The sequencer chooses four ALU threads (two from each bank), a vertex cache and a fetch thread to execute, and executes all of the instructions in a block before looking for a new clause of the same type. Two ALU threads are executed interleaved to hide the ALU latency. The arbitrator will give priority to older threads. There are two separate reservation stations, one for pixel vectors and one for vertices vectors. This way a pixel can pass a vertex and a vertex can pass a pixel.

There are also 2 separate ALU banks from which the SQ picks the ALU threads to be executed in parallel.

To support the shader pipe the sequencer also contains the shader instruction store, constant store, control flow constants and texture state. The height shader pipes also execute the same two instructions thus there is only one sequencer for the whole chip but it issues 2 instructions every four clocks.

The sequencer first arbitrates between vectors of 64 vertices that arrive directly from primitive assembly and vectors of 16 quads (64 pixels) that are generated in the scan converter.

The vertex or pixel program specifies how many GPRs it needs to execute. The sequencer will not start the next vector until the needed space is available in the GPRs.

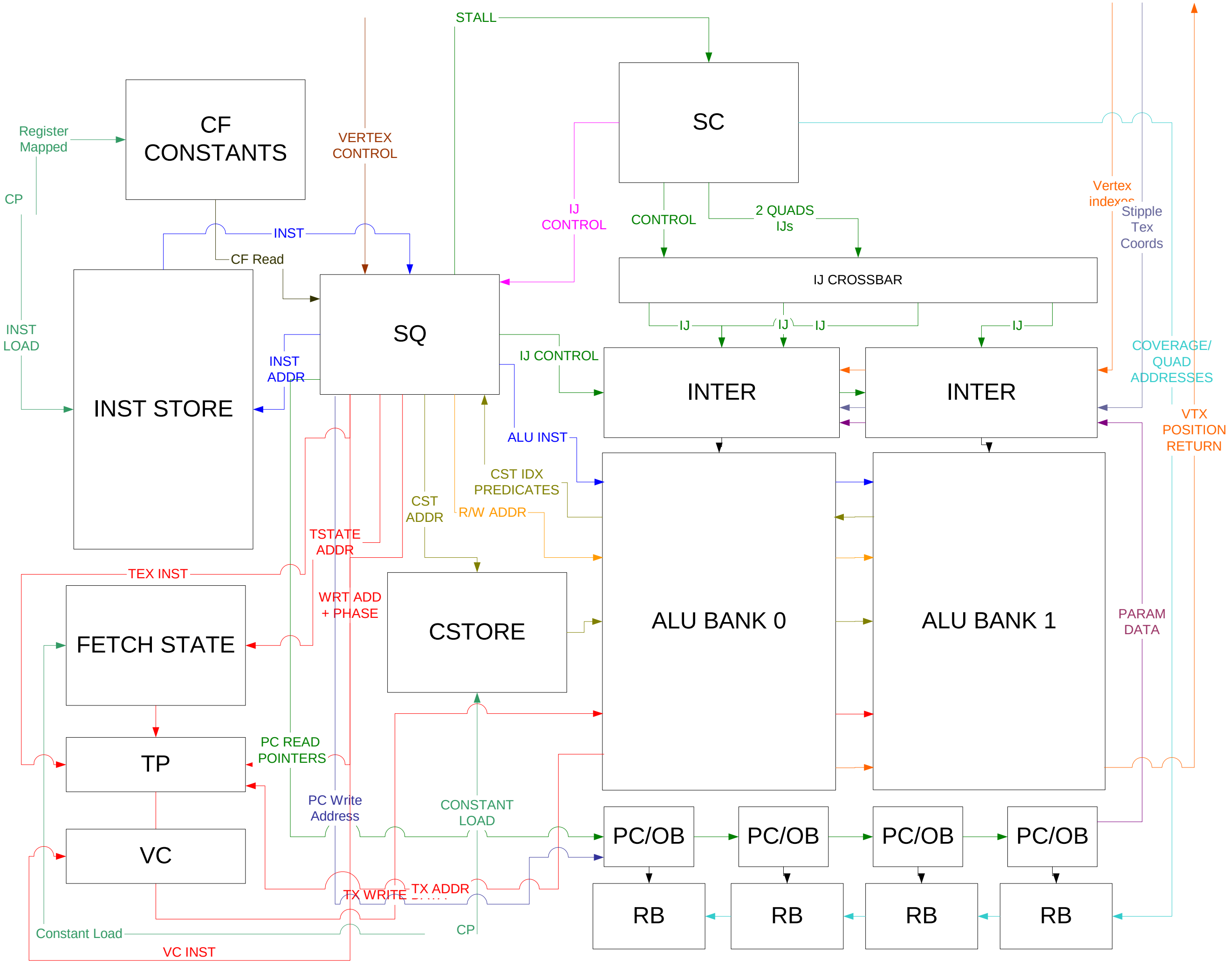


Figure 1: General Sequencer overview

1.1 Top Level Block Diagram

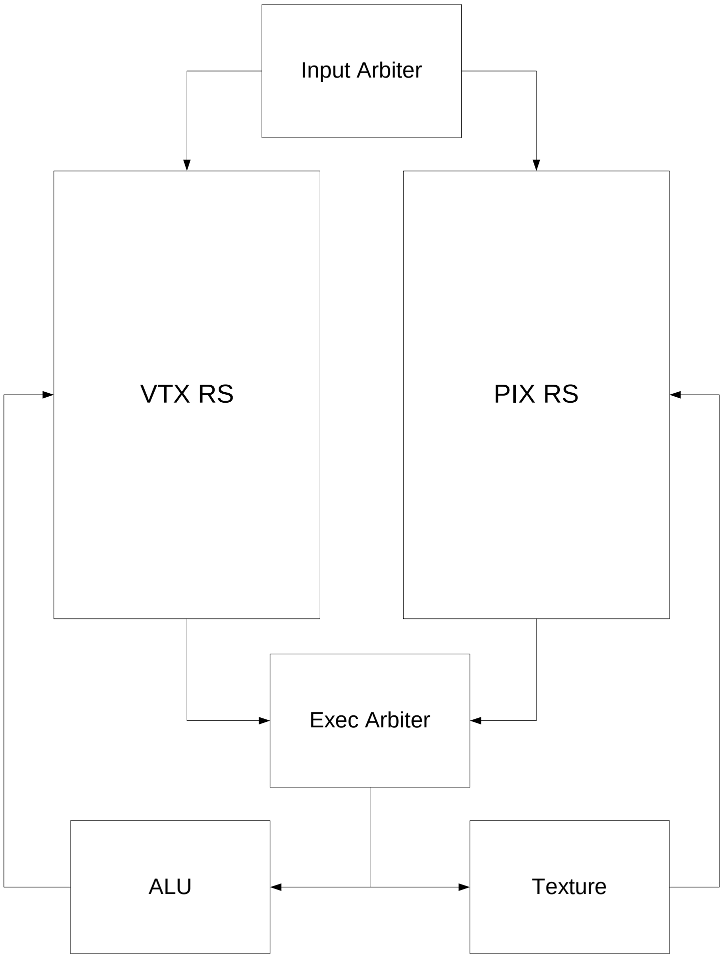


Figure 2: Reservation stations and arbiters

Under this new scheme, the sequencer (SQ) will only use one global state management machine per vector type (pixel, vertex) that we call the reservation station (RS).

1.2 Data Flow graph (SP)

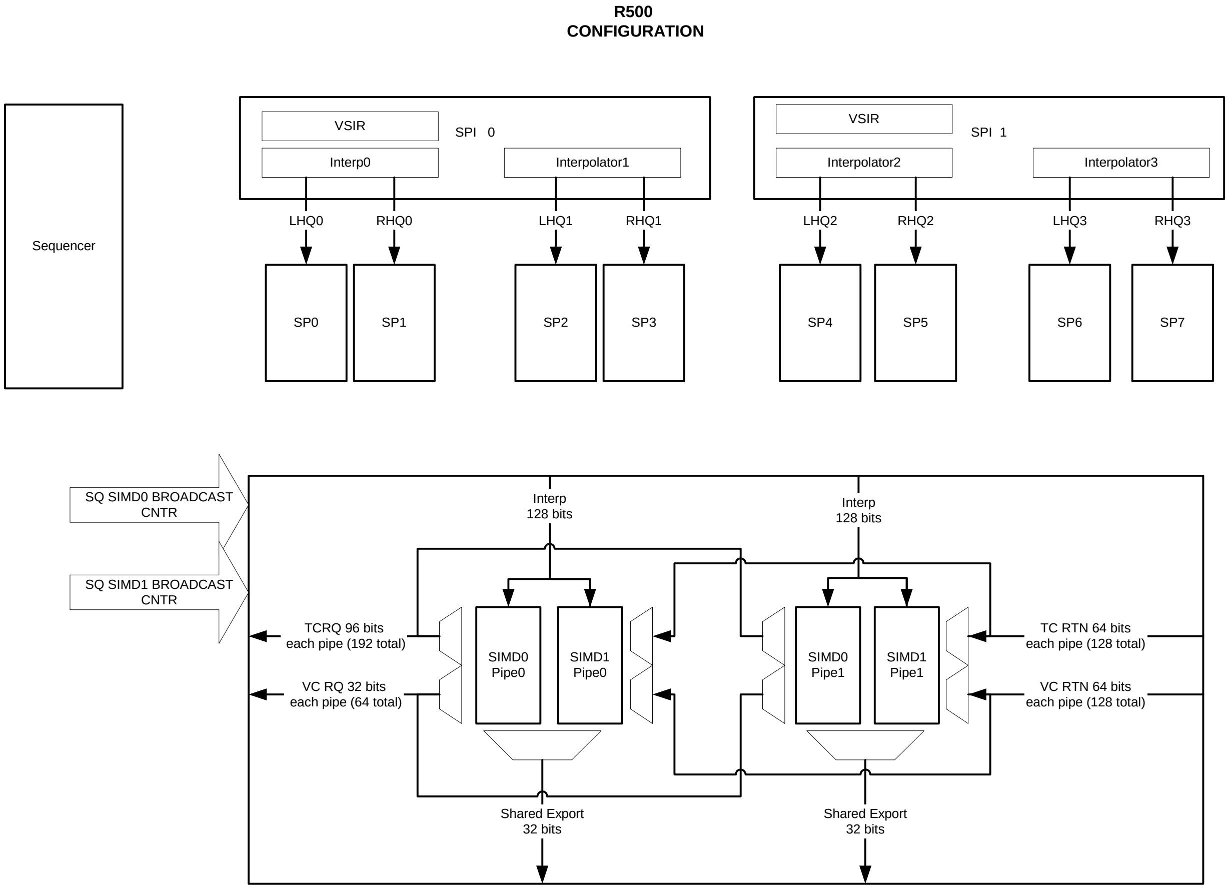


Figure 3: The shader Pipe

1.3 Control Graph

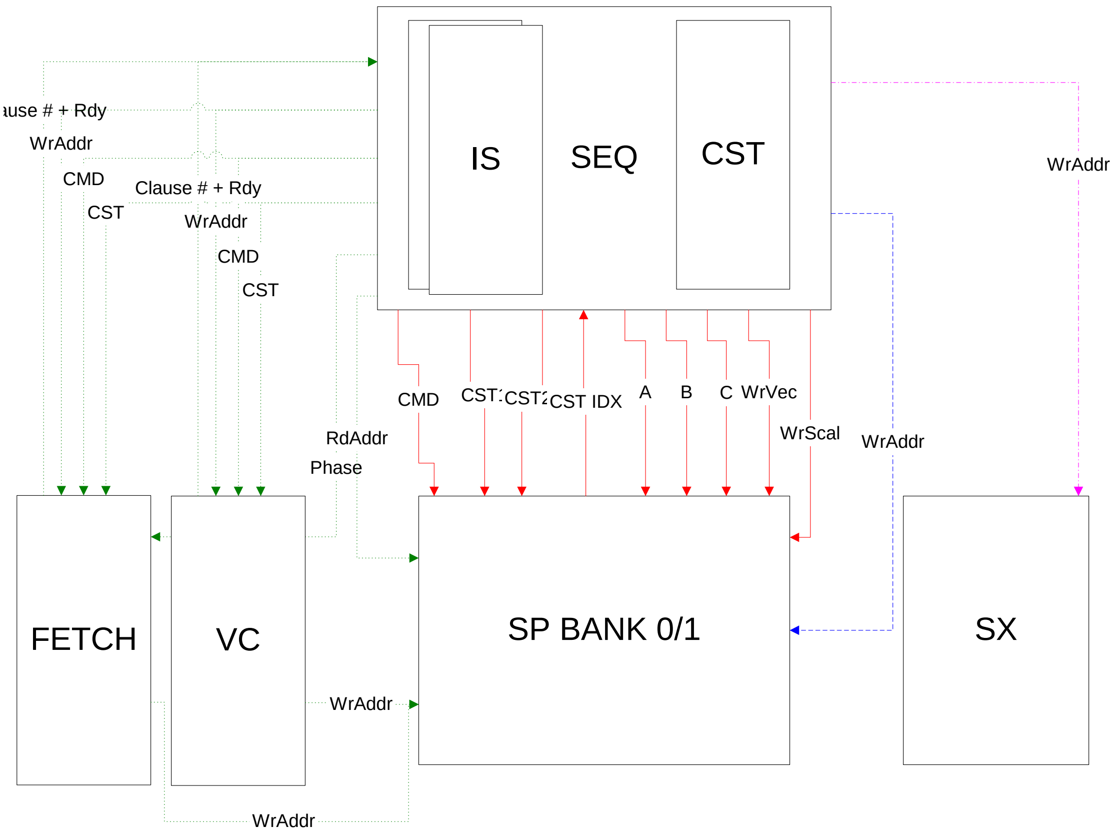


Figure 4: Sequencer Control interfaces

In green is represented the Fetch control interface, in red the ALU control interface, in blue the Interpolated/Vector control interface and in purple is the output file control interface.

2. Interpolated data bus

The interpolators contain an IJ buffer to pack the information as much as possible before writing it to the register file.

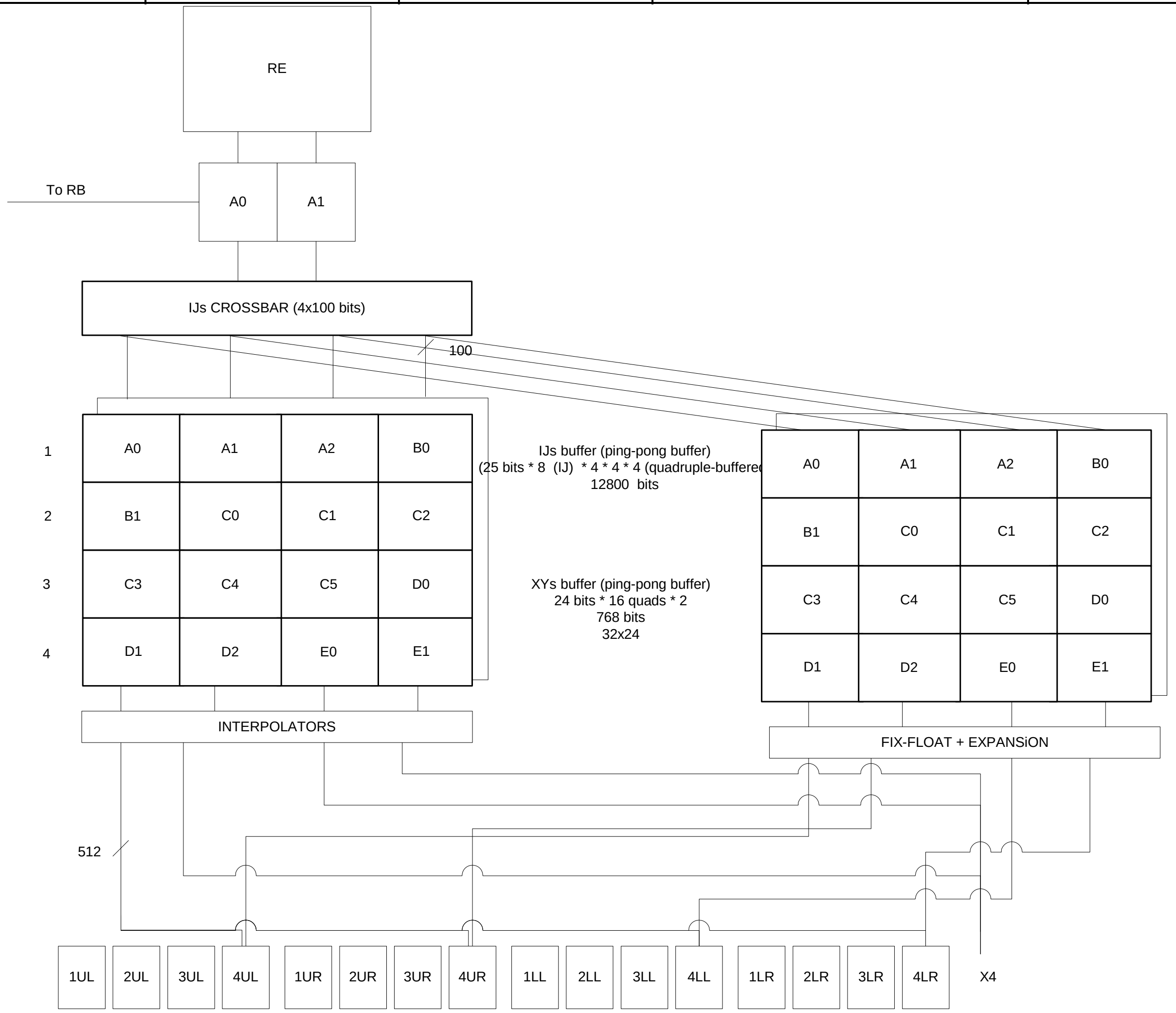


Figure 5: Interpolation buffers

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21	T22	T23
											WRITES													
SP 0	A0	A0	XY A0	B1	B1	XY B1	C3	C3	XY C3				D1	D1	XY D1									
SP 1	A1	A1	XY A1				C0	C0	XY C0	C4	C4	XY C4	D2	D2	XY D2									
SP 2	A2	A2	XY A2				C1	C1	XY C1	C5	C5	XY C5				E0	E0	XY E0						
SP 3				B0	B0	XY B0	C2	C2	XY C2				D0	D0	XY D0	E1	E1	XY E1						
											READS													
SP 0	XY 0-3	XY 16-19	XY 32-35	XY 48-51	A0	B1	C3	D1					A0	B1	C3	D1					V 0-3	V 16-19	V 32-35	V 48-51
SP 1	XY 4-7	XY 20-23	XY 36-39	XY 52-55	A1		C4	D2		C0			A1		C4	D2		C0			V 4-7	V 20-23	V 36-39	V 52-55
SP 2	XY 8-11	XY 24-27	XY 40-43	XY 56-59	A2		C5			C1		E0	A2		C5			C1		E0	V 8-11	V 24-27	V 40-43	V 56-59
SP 3	XY 12-15	XY 28-31	XY 44-47	XY 60-63					B0	C2	D0	E1					B0	C2	D0	E1	V 12-15	V 28-31	V 44-47	V 60-63
	XY				P1								P2								VTX			

Figure 6: Interpolation timing diagram

Above is an example of a tile the sequencer might receive from the SC. The write side is how the data get stacked into the XY and IJ buffers, the read side is how the data is passed to the GPRs. The IJ information is packed in the IJ buffer 4 quads at a time or two clocks. The sequencer allows at any given time as many as four quads to interpolate a parameter. They all have to come from the same primitive. Then the sequencer controls the write mask to the GPRs to write the valid data in.

3. Instruction Store

There is going to be two instruction stores for the whole chip. They will each contain 4096 instructions of 96 bits each.

They will be 1 port memories; Ports are allocated in this fashion (but not necessarily in this order):

ALU 0 SIMD0 CF
ALU 0 SIMD0
ALU 1 SIMD0 CF
ALU 1 SIMD0
Fetch CF
Fetch
VC CF
VC

ALU 0 SIMD1 CF
ALU 0 SIMD1
ALU 1 SIMD1 CF
ALU 1 SIMD1
Fetch CF
Fetch
VC CF
VC

Fetch and VC can steal one another's ports with stated resource having priority over its port (this is not really necessary for the R500 but will be for any derivative part because there will only be one instruction store).

Writes are opportunistic.

The instruction store is loaded by the CP thru the register mapped registers.

The VS_BASE and PS_BASE context registers are used to specify for each context where its shader is in the instruction memory.

For the Real time commands the story is quite the same but for some small differences. There are no wrap-around points for real time so the driver must be careful not to overwrite regular shader data. The shared code (shared subroutines) uses the same path as real time.

4. Sequencer Instructions

All control flow instructions instructions are handled by the sequencer only. The ALUs will perform NOPs during this time (MOV PV,PV, PS,PS) if they have nothing else to do.

5. Constant Stores

5.1 Memory organizations

A likely size for the ALU constant store is 1024x128 bits. The read BW from the ALU constant store is 128 bits/clock and the write bandwidth is 32 bits/clock (directed by the CP bus size not by memory ports).

The maximum logical size of the constant store for a given shader is 256 constants. Or 512 for the pixel/vertex shader pair. The size of the re-mapping table is 128 lines (each line addresses 4 constants). The write granularity is 4 constants or 512 bits. It takes 16 clocks to write the four constants. Real time requires 256 lines in the physical memory (this is physically register mapped).

There will be two of those memories and two of each remapping read memories.

The texture state is also kept in a similar memory. The size of this memory is 320x96 bits (128 texture states for regular mode, 32 states for RT). The memory thus holds 128 texture states (192 bits per state). The logical size exposes 32 different states total, which are going to be shared between the pixel and the vertex shader. The size of the re-mapping table to for the texture state memory is 32 lines (each line addresses 1 texture state lines in the real memory). The CP write granularity is 1 texture state lines (or 192 bits). The driver sends 512 bits but the CP ignores the top 320 bits. It thus takes 6 clocks to write the texture state. Real time requires 32 lines in the physical memory (this is physically register mapped).

The control flow constant memory doesn't sit behind a renaming table. It is register mapped and thus the driver must reload its content each time there is a change in the control flow constants. Its size is 320*32 because it must hold 8 copies of the 32 dwords of control flow constants and the loop construct constants must be aligned.

The constant re-mapping tables for texture state and ALU constants are logically register mapped for regular mode and physically register mapped for RT operation.

5.2 Management of the Control Flow Constants

The control flow constants are register mapped, thus the CP writes to the according register to set the constant, the SQ decodes the address and writes to the block pointed by its current base pointer (CF_WR_BASE). On the read side, one level of indirection is used. A register (SQ_CONTEXT_MISC.CF_RD_BASE) keeps the current base pointer to the control flow block. This register is copied whenever there is a state change. Should the CP write to CF after the state change, the base register is updated with the (current pointer number +1)% number of states. This way, if the CP doesn't write to CF the state is going to use the previous CF constants.

5.3 Management of the re-mapping tables

5.3.1 *R400 Constant management*

The sequencer is responsible to manage two re-mapping tables (one for the constant store and one for the texture state). On a state change (by the driver), the sequencer will broadside copy the contents of its re-mapping tables to a new one. We have 8 different re-mapping tables we can use concurrently.

The constant memory update will be incremental, the driver only need to update the constants that actually changed between the two state changes.

For this model to work in its simplest form, the requirement is that the physical memory MUST be at least twice as large as the logical address space + the space allocated for Real Time. In our case, since the logical address space is 512 and the reserved RT space can be up to 256 entries, the memory must be of sizes 1280 and above. Similarly the size of the texture store must be of 32*2+32 = 96 entries and above.

5.3.2 *Dirty bits*

Two sets of dirty bits will be maintained per logical address. The first one will be set to zero on reset and set when the logical address is addressed. The second one will be set to zero whenever a new context is written and set for each address written while in this context. The reset dirty is not set, then writing to that logical address will not require de-allocation of whatever address stored in the renaming table. If it is set and the context dirty is not set, then the physical address store needs to be de-allocated and a new physical address is necessary to store the incoming data. If they are both set, then the data will be written into the physical address held in the renaming for the current logical address. No de-allocation or allocation takes place. This will happen when the driver does a set constant twice to the same logical address between context changes. NOTE: It is important to detect and prevent this, failure to do it will allow multiple writes to allocate all physical memory and thus hang because a context will not fit for rendering to start and thus free up space.

5.3.3 *Free List Block*

A free list block that would consist of a counter (called the IFC or Initial Free Counter) that would reset to zero and incremented every time a chunk of physical memory is used until they have all been used once. This counter would be checked each time a physical block is needed, and if the original ones have not been used up, us a new one, else check the free list for an available physical block address. The count is the physical address for when getting a chunk from the counter.
Storage of a free list big enough to store all physical block addresses.
Maintain three pointers for the free list that are reset to zero. The first one we will call write_ptr. This pointer will identify the next location to write the physical address of a block to be de-allocated. Note: we can never free more

physical memory locations than we have. Once recording address the pointer will be incremented to walk the free list like a ring.
The second pointer will be called stop_ptr. The stop_ptr pointer will be advanced by the number of address chunks de-allocates when a context finishes. The address between the stop_ptr and write_ptr cannot be reused because they are still in use. But as soon as the context using then is dismissed the stop_ptr will be advanced.
The third pointer will be called read_ptr. This pointer will point will point to the next address that can be used for allocation as long as the read_ptr does not equal the stop_ptr and the IFC is at its maximum count.

5.3.4 *De-allocate Block*

This block will maintain a free physical address block count for each context. While in current context, a count shall be maintained specifying how many blocks were written into the free list at the write_ptr pointer. This count will be reset upon reset or when this context is active on the back and different than the previous context. It is actually a count of blocks in the previous context that will no longer be used. This count will be used to advance the write_ptr pointer to make available the set of physical blocks freed when the previous context was done. This allows the discard or de-allocation of any number of blocks in one clock.

5.3.5 *Operation of Incremental model*

The basic operation of the model would start with the write_ptr, stop_ptr, read_ptr pointers in the free list set to zero and the free list counter is set to zero. Also all the dirty bits and the previous context will be initialized to zero. When the first set constants happen, the reset dirty bit will not be set, so we will allocate a physical location from the free list counter because its not at the max value. The data will be written into physical address zero. Both the additional copy of the renaming table and the context zeros of the big renaming table will be updated for the logical address that was written by set start with physical address of 0. This process will be repeated for any logical address that are not dirty until the context changes. If a logical address is hit that has its dirty bits set while in the same context, both dirty bits would be set, so the new data will be over-written to the last physical address assigned for this logical address. When the first draw command of the context is detected, the previous context stored in the additional renaming table will be copied to the larger renaming table in the current (new) context location. Then the set constant logical address with be loaded with a new physical address during the copy and if the reset dirty was set, the physical address it replaced in the renaming table would be entered at the write_ptr pointer location on the free list and the write_ptr will be incremented. The de-allocation counter for the previous context (eight) will be incremented. This as set states come in for this context one of the following will happen:

1.) No dirty bits are set for the logical address being updated. A line will be allocated of the free-list counter or the free list at read_ptr pointer if read_ptr != to stop_ptr .
2.) Reset dirty set and Context dirty not set. A new physical address is allocated, the physical address in the renaming table is put on the free list at write_ptr and it is incremented along with the de-allocate counter for the last context.
3.) Context dirty is set then the data will be written into the physical address specified by the logical address.

This process will continue as long as set states arrive. This block will provide backpressure to the CP whenever he has not free list entries available (counter at max and stop_ptr == read_ptr). The command stream will keep a count of contexts of constants in use and prevent more than max constants contexts from being sent.

Whenever a draw packet arrives, the content of the re-mapping table is written to the correct re-mapping table for the context number. Also if the next context uses less constants than the current one all exceeding lines are moved to the free list to be de-allocated later. This happens in parallel with the writing of the re-mapping table to the correct memory.

Now preferable when the constant context leaves the last ALU clause it will be sent to this block and compared with the previous context that left. (Init to zero) If they differ than the older context will no longer be referenced and thus can be de-allocated in the physical memory. This is accomplished by adding the number of blocks freed this context to the stop_ptr pointer. This will make all the physical addresses used by this context available to the read_ptr allocate pointer for future allocation.

This device allows representation of multiple contexts of constants data with N copies of the logical address space. It also allows the second context to be represented as the first set plus some new additional data by just storing the delta's. It allows memory to be efficiently used and when the constants updates are small it can store multiple context. However, if the updates are large, less contexts will be stored and potentially performance will be degraded. Although it will still perform as well as a ring could in this case.

5.4 Constant Store Indexing

In order to do constant store indexing, the sequencer must be loaded first with the indexes (that come from the GPRs). There are 144 wires from the exit of the SP to the sequencer (9 bits pointers x 16 vertexes/clock).

```
MOVA  R1.X,R2.X        // Loads the sequencer with the content of R2.X, also copies the content of R2.X into R1.X
ADD   R3,R4,C0[R2.X]// Uses the state from the sequencer to add R4 to C0[R2.X] into R3
```

Note that we don't really care about what is in the brackets because we use the state from the MOVA instruction. R2.X is just written again for the sake of simplicity and coherency.

The storage needed in the sequencer in order to support this feature is 2*64*9 bits = 1152 bits.

The address register is a signed integer, which ranges from –256 to 255.

5.5 Real Time Commands

The real time commands constants are written by the CP using the register mapped registers allocated for RT. It works is the same way than when dealing with regular constant loads BUT in this case the CP is not sending a logical address but rather a physical address and the reads are not passing thru the re-mapping table but are directly read from the memory. The boundary between the two zones is defined by the CONST_EO_RT control register. Similarly, for the fetch state, the boundary between the two zones is defined by the TSTATE_EO_RT control register.

5.6 Constant Waterfalling

In order to have a reasonable performance in the case of constant store indexing using the address register, we are going to have the possibility of using the physical memory port for read only. This way we can read 1 constant per clock and thus have a worst-case waterfall mode of 1 vertex per clock. There is a small synchronization issue related with this as we need for the SQ to make sure that the constants where actually written to memory (not only sent to the sequencer) before it can allow the first vector of pixels or vertices of the state to go thru the ALUs. To do so, the sequencer keeps 8 bits (one per render state) and sets the bits whenever the last render state is written to memory and clears the bit whenever a state is freed.

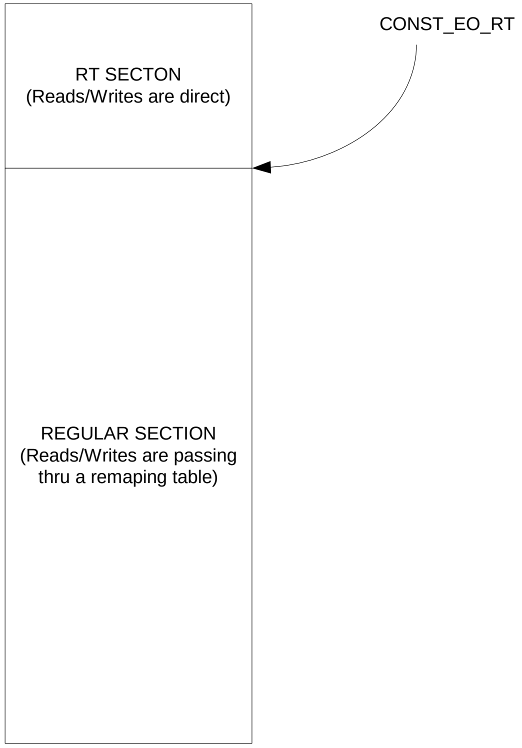

CONST_EO_RT
RT SECTON
(Reads/Writes are direct)
REGULAR SECTION
(Reads/Writes are passing thru a remaping table)

Figure 7: The Constant store

6. Looping and Branches

Loops and branches are planned to be supported and will have to be dealt with at the sequencer level. We plan on supporting constant loops and branches using a control program.

6.1 The controlling state.

The R400 controling state consists of:

Boolean[255:0]
Loop_count[7:0][31:0]
Loop_Start[7:0][31:0]
Loop_Step[7:0][31:0]

That is 256 Booleans and 32 loops.

We have a stack of 4 elements for nested calls of subroutines and 4 loop counters to allow for nested loops.

This state is available on a per shader program basis.

6.2 The Control Flow Program

We'd like to be able to code up a program of the form:

```
1:      Loop
2:      Exec    TexFetch
```

3:                TexFetch
4:                ALU
5:                ALU
6:                TexFetch
7:        End Loop
8:        ALU Export

But realize that 3: may be dependent on 2: and 4: is almost certainly dependent on 2: and 3:. Without clausing, these dependencies need to be expressed in the Control Flow instructions. Additionally, without separate 'texture clauses' and 'ALU clauses' we need to know which instructions to dispatch to the Texture Unit and which to the ALU unit. This information will be encapsulated in the flow control instructions.

Each control flow instruction will contain 2 bits of information for each (non-control flow) instruction:

a) ALU or Texture
b) Serialize Execution

(b) would force the thread to stop execution at this point (before the instruction is executed) and wait until all textures have been fetched. Given the allocation of reserved bits, this would mean that the count of an 'Exec' instruction would be limited to about 8 (non-control-flow) instructions. If more than this were needed, a second Exec (with the same conditions) would be issued.

Another function that relies upon 'clauses' is allocation and order of execution. We need to assure that pixels and vertices are exported in the correct order (even if not all execution is ordered) and that space in the output buffers are allocated in order. Additionally data can't be exported until space is allocated. A new control flow instruction:

Alloc <buffer select -- position,parameter, pixel or vertex memory. And the size required>.

would be created to mark where such allocation needs to be done. To assure allocation is done in order, the actual allocation for a given thread can not be performed unless the equivalent allocation for all previous threads is already completed. The implementation would also assure that execution of instruction(s) following the serialization due to the Alloc will occur in order -- at least until the next serialization or change from ALU to Texture. In most cases this will allow the exports to occur without any further synchronization. Only 'final' allocations or position allocations are guaranteed to be ordered. Because strict ordering is required for pixels, parameters and positions, this implies only a single alloc for these structures. Vertex exports to memory do not require ordering during allocation and so multiple 'allocs' may be done.

6.2.1 *Control flow instructions table*

Here is the revised control flow instruction set.

Note that whenever a field is marked as RESERVED, it is assumed that all the bits of the field are cleared (0).

NOP		
47 … 44	43	42 … 0
0000	Addressing	RESERVED

This is a regular NOP.

Execute						
47 … 44	43	40 … 34	33 … 28	27 …16	15…12	11 … 0
0001	Addressing	RESERVED	Vertex Cache	Instructions type + serialize (6 instructions)	Count	Exec Address

Execute_End						
47 … 44	43	40 … 34	33 … 28	27 …16	15…12	11 … 0
0010	Addressing	RESERVED	Vertex Cache	Instructions type + serialize (6 instructions)	Count	Exec Address

Execute up to 6 instructions at the specified address in the instruction memory. The Instruction type field tells the sequencer the type of the instruction (LSB) (1 = Texture, 0 = ALU and whether to serialize or not the execution (MSB) (1 = Serialize, 0 = Non-Serialized). If the corresponding VC bit is set then VC is used instead of TP/ALU. If Execute_End this is the last execution block of the shader program.

Vertex Cache	Serialize	Instruction Type (Resource)	
0	0	0	: ALU instruction, not yielding
0	0	1	: ALU instruction, yielding
0	1	0	: Texture instruction, not yielding
0	1	1	: Texture instruction, yielding
1	0	0	: Vertex cache instruction, not yielding
1	0	1	: Vertex cache instruction, yielding
1	1	0	: Vertex cache instruction, not yielding
1	1	1	: Vertex cache instruction, yielding

Conditional_Execute							
47 … 44	43	42	41 … 34	33…28	27…16	15 …12	11 … 0
0011	Addressing	Condition	Boolean address	Vertex Cache	Instructions type + serialize (6 instructions)	Count	Exec Address

Conditional_Execute_End							
47 … 44	43	42	41 … 34	33…28	27…16	15 …12	11 … 0
0100	Addressing	Condition	Boolean address	Vertex Cache	Instructions type + serialize (6 instructions)	Count	Exec Address

If the specified Boolean (8 bits can address 256 Booleans) meets the specified condition then execute the specified instructions (up to 9 instructions). If the condition is not met, we go on to the next control flow instruction. If Conditional_Execute_End and the condition is met, this is the last execution block of the shader program.

Conditional_Execute_Predicates								
47 … 44	43	42	41 … 36	35 … 34	33…28	27…16	15…12	11 … 0
0101	Addressing	Condition	RESERVED	Predicate vector	Vertex Cache	Instructions type + serialize (6 instructions)	Count	Exec Address

Conditional_Execute_Predicates_End								
47 … 44	43	42	41 … 36	35 … 34	33…28	27…16	15…12	11 … 0
0110	Addressing	Condition	RESERVED	Predicate vector	Vertex Cache	Instructions type + serialize (6 instructions)	Count	Exec Address

Check the AND/OR of all current predicate bits. If AND/OR matches the condition execute the specified number of instructions. We need to AND/OR this with the kill mask in order not to consider the pixels that aren't valid. If the condition is not met, we go on to the next control flow instruction. If Conditional_Execute_Predicates_End and the condition is met, this is the last execution block of the shader program.

Conditional_Execute_Predicates_No_Stall								
47 … 44	43	42	41 … 36	35 … 34	33…28	27…16	15…12	11 … 0
1101	Addressing	Condition	RESERVED	Predicate vector	Vertex Cache	Instructions type + serialize (6 instructions)	Count	Exec Address

Conditional_Execute_Predicates_No_Stall_End								
47 … 44	43	42	41 … 36	35 … 34	33…28	27…16	15…12	11 … 0
1110	Addressing	Condition	RESERVED	Predicate vector	Vertex Cache	Instructions type + serialize (6 instructions)	Count	Exec Address

Same as Conditionnal_Execute_Predicates but the SQ is not going to wait for the predicate vector to be updated. You can only set this in the compiler if you know that the predicate set is only a refinement of the current one (like a nested if) because the optimization would still work.

Loop_Start					
47 … 44	43	42 … 21	20 … 16	15…13	12 … 0
0111	Addressing	RESERVED	loop ID	RESERVED	Jump address

Loop Start. Compares the loop iterator with the end value. If loop condition not met jump to the address. Forward jump only. Also computes the index value. The loop id must match between the start to end, and also indicates which control flow constants should be used with the loop.

Loop_End									
47 …44	43	42	41… 36	35…34	33… 22	21	20 … 16	15…13	12 … 0
1000	Addressing	Cond	RESERVED	Predicate Vector	RESERVED	Pred break	loop ID	RESERVED	start address

Loop end. Increments the counter by one, compares the loop count with the end value. If loop condition met, continue, else, jump BACK to the start of the loop. If predicate break != 0, then compares predicate vector n (specified by predicate Vector) to condition. If all bits meet condition then break the loop.

The way this is described does not prevent nested loops, and the inclusion of the loop id make this easy to do.

Conditionnal_Call						
47 … 44	43	42	41 … 34	33 … 14	13	12 … 0
1001	Addressing	Condition	Boolean address	RESERVED	Force Call	Jump address

If the condition is met, jumps to the specified address and pushes the control flow program counter on the stack. If force call is set the condition is ignored and the call is made always.

Return		
47 … 44	43	42 … 0
1010	Addressing	RESERVED

Pops the topmost address from the stack and jumps to that address. If nothing is on the stack, the program will just continue to the next instruction.

Conditionnal_Jump							
47 … 44	43	42	41… 34	33	32 … 14	13	12 … 0
1011	Addressing	Condition	Boolean address	FW only	RESERVED	Force Jump	Jump address

If force jump is set the condition is ignored and the jump is made always. If FW only is set then only forward jumps are allowed.

Allocate					
47 … 44	43	42…41	40	39 … 3	2…0
1100	Debug	Buffer Select	No Serial	RESERVED	Size

Buffer Select takes a value of the following:
01 – position export (ordered export)
10 – parameter cache or pixel export (ordered export)
11 – pass thru (out of order exports).

Size field is only used to reserve space in the export buffer for pass thru exports. Valid values are 1 (1 line) thru 9 (9 lines). It should be determined by the compiler/assembler by taking max index used +1.

If debug is set this is a debug alloc (ignore if debug DB_ON register is set to off).

By default the serial bit is set on an alloc. If the No Serial bit is asserted then the serial bit won't be set in the SQ.

6.3 Implementation

The envisioned implementation has a buffer that maintains the state of each thread. A thread lives in a given location in the buffer during its entire life, but the buffer has FIFO qualities in that threads leave in the order that they enter. Actually two buffers are maintained -- one for Vertices and one for Pixels. The intended implementation would allow for:

16 entries for vertices
48 entries for pixels.

From each buffer, arbitration logic attempts to select 1 thread for the texture unit and 2 (interleaved) thread for the ALU unit. Once a thread is selected it is read out of the buffer, marked as invalid, and submitted to appropriate execution unit. It is returned to the buffer (at the same place) with its status updated once all possible sequential instructions have been executed. A switch from ALU to TEX or visa-versa or a Serialize_Execution modifier forces the thread to be returned to the buffer.

Each entry in the buffer will be stored across two physical pieces of memory - most bits will be stored in a 1 read port device. Only bits needed for thread arbitration will be stored in a highly multi-ported structure. The bits kept in the 1 read port device will be termed 'state'. The bits kept in the multi-read ported device will be termed 'status'.

'State Bits' needed include:

1. Control Flow Instruction Pointer (13 bits),
2. Execution Count Marker 4 bits),
3. Loop Iterators (4x9 bits),
4. Loop Counters (4x9 bits),
5. Call return pointers (4x13 bits),
6. Predicate Bits (64 bits),
7. Export ID (4 bits),
8. Parameter Cache base Ptr (7 bits),
9. GPR Base Ptr (8 bits),
10. Context Ptr (3 bits).
11. LOD corrections (6x16 bits)
12. Valid bits (64 bits)
13. RT (1 bit) Signifies that this thread is a Real Time thread. This bit must be sent to the Constant store state machine when reading it.

Absent from this list are 'Index' pointers. These are costly enough that I'm presuming that they are instead stored in the GPRs. The first seven fields above (Control Flow Ptr, Execution Count, Loop Counts, call return ptrs, Predicate bits, PC base ptr and export ID) are updated every time the thread is returned to the buffer based on how much progress has been mode on thread execution. GPR Base Ptr, Context Ptr and LOD corrections are unchanged throughout execution of the thread.

'Status Bits' needed include:

- Valid Thread
- ALU engine needed
- Texture engine needed
- VC engine needed
- Texture Reads are outstanding
- VC Reads are outstanding
- Alu bank (0/1)
- Waiting on Texture Read to Complete
- Allocation Wait (2 bits)
- 00 – No allocation needed
- 01 – Position export allocation needed (ordered export)
- 10 – Parameter or pixel export needed (ordered export)
- 11 – pass thru (out of order export)
- Allocation Size (4 bits)
- Position Allocated
- Mem/Color Allocated
- First thread of a new context
- Event thread (NULL thread that needs to trickle down the pipe)
- Last (1 bit)
- Pulse SX (1 bit)

All of the above fields from all of the entries go into the arbitration circuitry. The arbitration circuitry will select a winner for both the Texture Engine and for the ALU engine. There are actually two sets of arbitration -- one for pixels and one for vertices. A final selection is then done between the two. But the rest of this implementation summary only considers the 'first' level selection which is similar for both pixels and vertices.

Texture arbitration requires no allocation or ordering so it is purely based on selecting the 'oldest' thread that requires the Texture Engine.

ALU arbitration is a little more complicated. First, only threads where either of Texture_Reads_outstanding or Waiting_on_Texture_Read_to_Complete are '0' are considered. Then if Allocation_Wait is active, these threads are further filtered based on whether space is available. If the allocation is position allocation, then the thread is only considered if all 'older' threads have already done their position allocation (position allocated bits set). If the allocation is parameter or pixel allocation, then the thread is only considered if it is the oldest thread. Also a thread is not considered if it is a parameter or pixel or position allocation, has its First_thread_of_a_new_context bit set and would cause ALU interleaving with another thread performing the same parameter or pixel or position allocation. Finally the 'oldest' of the threads that pass through the above filters is selected. If the thread needed to allocate, then at this time the allocation is done, based on Allocation_Size. If a thread has its “last” bit set, then it is also removed from the buffer, never to return.

If I now redefine 'clauses' to mean 'how many times the thread is removed from the thread buffer for the purpose of exection by either the ALU or Texture engine', then the minimum number of clauses needed is 2 -- one to perform the allocation for exports (execution automatically halts after an 'Alloc' instruction) (but doesn't performs the actual allocation) and one for the actual ALU/export instructions. As the 'Alloc' instruction could be part of a texture clause (presumably the final instruction in such a clause), a thread could still execute in this minimal number of 2 clauses, even if it involved texture fetching.

The Texture_Reads_Outstanding and VC_reads_Outstanding bits tell the SQ that a texture or VC read is outstanding. In this case, if we encounter a serial bit we need to wait until both resources are free (pending = 0) in order to proceed.

6.4 Data dependant predicate instructions

Data dependant conditionals will be supported in the R400. The only way we plan to support those is by supporting three vector/scalar predicate operations of the form:

PRED_SETE_PUSH - similar to SETE except that the result is 'exported' to the sequencer.
PRED_SETNE_PUSH - similar to SETNE except that the result is 'exported' to the sequencer.
PRED_SETGT_PUSH - similar to SETGT except that the result is 'exported' to the sequencer
PRED_SETGTE_PUSH - similar to SETGTE except that the result is 'exported' to the sequencer

For the scalar operations only we will also support the two following instructions:
PRED_SETE
PRED_SETNE
PRED_SETGT
PRED_SET_INV
PRED_SET_POP
PRED_SET_CLR
PRED_SET_RESTORE

Details about actual implementation of these opcodes are in the shader pipe architectural spec.

The export is a single bit - 1 or 0 that is sent using the same data path as the MOVA instruction. The sequencer will maintain 1 set of 64 bits predicate vectors (in fact 2 sets because we interleave two programs but only 1 will be exposed) and use it to control the write masking. This predicate is maintained across clause boundaries.

Then we have two conditional execute bits. The first bit is a conditional execute “on” bit and the second bit tells us if we execute on 1 or 0. For example, the instruction:

P0_ ADD_# R0,R1,R2

Is only going to write the result of the ADD into those GPRs whose predicate bit is 0. Alternatively, P1_ADD_# would only write the results to the GPRs whose predicate bit is set. The use of the P0 or P1 without precharging the sequencer with a PRED instruction is undefined.

6.5 HW Detection of PV,PS

Because of the control program, the compiler cannot detect statically dependant instructions. In the case of non-masked writes and subsequent reads the sequencer will insert uses of PV,PS as needed. This will be done by comparing the read address and the write address of consecutive instructions. For masked writes, the sequencer will insert detect wich channels to read from the GPRs and which ones to read from the PV/PS.

6.6 Register file indexing

Because we can have loops in fetch clause, we need to be able to index into the register file in order to retrieve the data created in a fetch clause loop and use it into an ALU clause. The instruction will include the base address for register indexing and the instruction will contain these controls:

Bit7	Bit 6	
0	0	'absolute register'
0	1	'relative register'
1	0	'previous vector'
1	1	'previous scalar'

In the case of an absolute register we just take the address as is. In the case of a relative register read we take the base address and we add to it the loop_index and this becomes our new address that we give to the shader pipe.

The sequencer is going to keep a loop index computed as such:

Index = Loop_iterator*Loop_step + Loop_start.

We loop until loop_iterator = loop_count. Loop_step is a signed value [-128…127]. The computed index value is a 10 bit counter that is also signed. Its real range is [-256,256]. The tenth bit is only there so that we can provide an out of range value to the "indexing logic" so that it knows when the provided index is out of range and thus can make the necessary arrangements.

6.7 Debugging the Shaders

In order to be able to debug the pixel/vertex shaders efficiently, we provide 2 methods.

6.7.1 *Method 1: Debugging registers*

Current plans are to expose 2 debugging, or error notification, registers:
1. address register where the first error occurred
2. count of the number of errors

The sequencer will detect the following groups of errors:
- count overflow
- constant indexing overflow
- register indexing overflow

Compiler recognizable errors:
- jump errors
    relative jump address > size of the control flow program
- call stack
    call with stack full
    return with stack empty

With all the other errors, program can continue to run, potentially to worst-case limits.

If indexing outside of the constant or the register range, causing an overflow error, the hardware is specified to return the value with an index of 0. This could be exploited to generate error tokens, by reserving and initializing the 0th register (or constant) for errors.

{ISSUE : Interrupt to the driver or not?}

6.7.2 *Method 2: Exporting the values in the GPRs*

1) The sequencer will have a debug active, count register and an address register for this mode.

Under the normal mode execution follows the normal course.

Under the debug mode it is assumed that the program is always exporting n debug vectors and that all other exports to the SX block (but for position) will be turned off (changed into NOPs) by the sequencer (even if they occur before the address stated by the ADDR debug register).

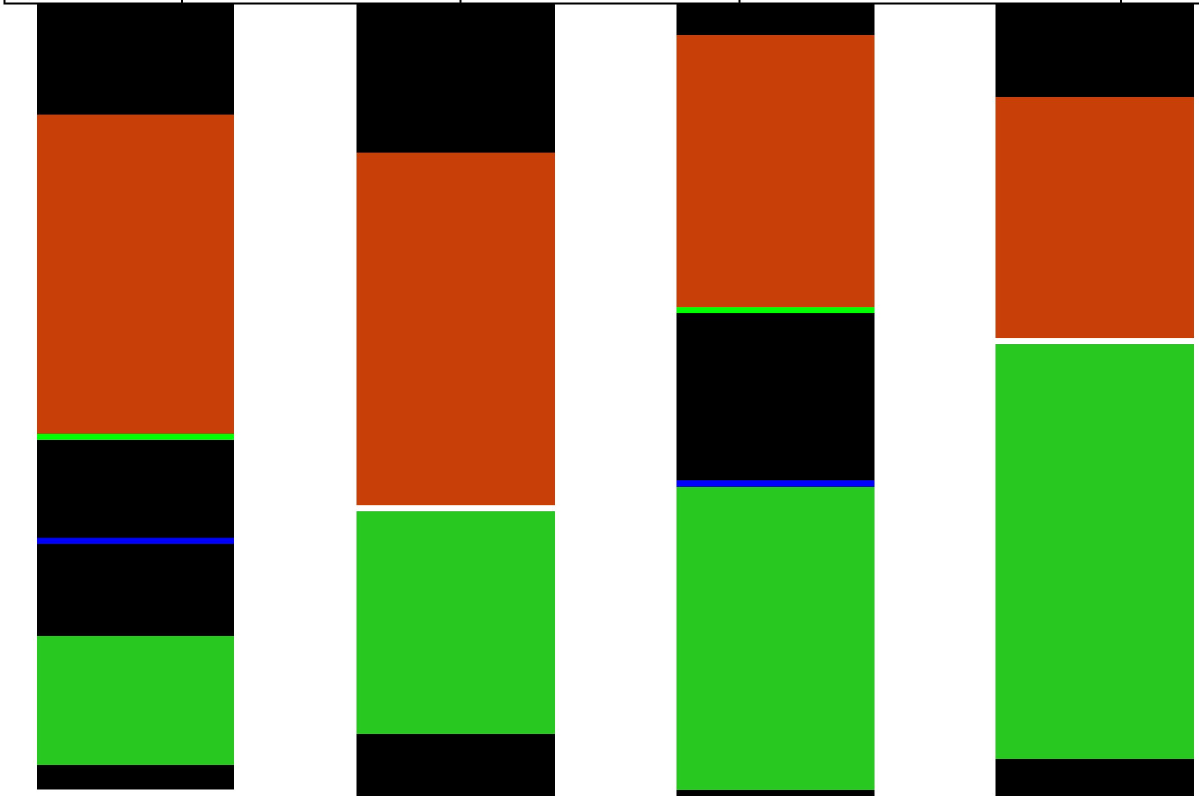
7. Pixel Kill Mask

A vector of 64 bits is kept by the sequencer per group of pixels/vertices. Its purpose is to optimize the texture fetch requests and allow the shader pipe to kill pixels using the following instructions:

MASK_SETE
MASK_SETNE
MASK_SETGT
MASK_SETGTE

8. Register file allocation

The register file allocation for vertices and pixels can either be static or dynamic. In both cases, the register file in managed using two round robins (one for pixels and one for vertices). In the dynamic case the boundary between pixels and vertices is allowed to move, in the static case it is fixed to 128-VERTEX_REG_SIZE for vertices and PIXEL_REG_SIZE for pixels.



Above is an example of how the algorithm works. Vertices come in from top to bottom; pixels come in from bottom to top. Vertices are in orange and pixels in green. The blue line is the tail of the vertices and the green line is the tail of the pixels. Thus anything between the two lines is shared. When pixels meets vertices the line turns white and the boundary is static until both vertices and pixels share the same “unallocated bubble”. Then the boundary is allowed to move again. The numbering of the GPRs starts from the bottom of the picture at index 0 and goes up to the top at index 127.

9. Fetch Arbitration

The fetch arbitration logic chooses one of the n potentially pending fetch clauses to be executed. The choice is made by looking at the Vs and Ps reservation stations and picking the first one ready to execute. Once chosen, the clause state machine will send one 2x2 fetch per clock (or 4 fetches in one clock every 4 clocks) until all the fetch instructions of the clause are sent. This means that there cannot be any dependencies between two fetches of the same clause.

The arbitrator will not wait for the fetches to return prior to selecting another clause for execution. The fetch pipe will be able to handle up to X(?) in flight fetches and thus there can be a fair number of active clauses waiting for their fetch return data.

10. VC Arbitration

The VC arbitration logic chooses one of the n potentially pending VC clauses to be executed. The choice is made by looking at the Vs and Ps reservation stations and picking the first one ready to execute. Once chosen, the clause state machine will send one 2x2 fetch per clock (or 4 fetches in one clock every 4 clocks) until all the fetch instructions of the clause are sent. This means that there cannot be any dependencies between two fetches of the same clause.

The arbitrator will not wait for the fetches to return prior to selecting another clause for execution. The VC pipe will be able to handle up to X(?) in flight VC fetches and thus there can be a fair number of active clauses waiting for their fetch return data.

11. ALU Arbitration

ALU arbitration proceeds in almost the same way than fetch arbitration. The ALU arbitration logic chooses one of the n potentially pending ALU clauses to be executed. The choice is made by looking at the Vs and Ps reservation stations and picking the first one ready to execute. There are two ALU arbiters, one for the even clocks and one for the odd clocks. For example, here is the sequencing of two interleaved ALU clauses (E and O stands for Even and Odd sets of 4 clocks):

Einst0 Oinst0 Einst1 Oinst1 Einst2 Oinst2 Einst0 Oinst3 Einst1 Oinst4 Einst2 Oinst0…
Proceeding this way hides the latency of 8 clocks of the ALUs. Also note that the interleaving also occurs across clause boundaries.

12. Handling Stalls

When the output file is full, the sequencer prevents the ALU arbitration logic from selecting the last clause (this way nothing can exit the shader pipe until there is place in the output file. If the packet is a vertex packet and the position buffer is full (POS_FULL) then the sequencer also prevents a thread from entering an exporting clause. The sequencer will set the OUT_FILE_FULL signal n clocks before the output file is actually full and thus the ALU arbiter will be able read this signal and act accordingly by not preventing exporting clauses to proceed.

12.1 SP stall conditions

12.1.1 *PS Stalls*

None.

12.1.2 *PV Stalls*

None.

13. Content of the reservation station FIFOs

The reservation FIFOs contain the state of the vector of pixels and vertices. We have two sets of those: one for pixels, and one for vertices. They contain 3 bits of Render State 7 bits for the base address of the GPRs, some bits for LOD correction and coverage mask information in order to fetch fetch for only valid pixels, the quad address.

14. The Output File

The output file is where pixels are put before they go to the RBs. The write BW to this store is 256 bits/clock. Just before this output file are staging registers with write BW 512 bits/clock and read BW 256 bits/clock. The staging registers are 4x128 (and there are 16 of those on the whole chip).

15. IJ Format

The IJ information sent by the PA is of this format on a per quad basis:

We have a vector of IJ's (one IJ per pixel at the centroid of the fragment or at the center of the pixel depending on the mode bit). All pixel's parameters are always interpolated at full 20x24 mantissa precision.

$$P0 = A + I(0) * (B - A) + J(0) * (C - A)$$
$$P1 = A + I(1) * (B - A) + J(1) * (C - A)$$
$$P2 = A + I(2) * (B - A) + J(2) * (C - A)$$
$$P3 = A + I(3) * (B - A) + J(3) * (C - A)$$

P0	P1
P2	P3

Multiplies (Full Precision): 8
Subtracts 19x24 (Parameters): 2
Adds: 8

FORMAT OF P's IJ : Mantissa 20 Exp 4 for I + Sign
Mantissa 20 Exp 4 for J + Sign

Total number of bits : 20*8 + 4*8 + 4*2 = 200.

All numbers are kept using the un-normalized floating point convention: if exponent is different than 0 the number is normalized if not, then the number is un-normalized. The maximum range for the IJs (Full precision) is +/- 1024.

15.1 Interpolation of constant attributes

Because of the floating point imprecision, we need to take special provisions if all the interpolated terms are the same or if two of the terms are the same.

16. Staging Registers

In order for the reuse of the vertices to be 14, the sequencer will have to re-order the data sent IN ORDER by the VGT for it to be aligned with the parameter cache memory arrangement. Given the following group of vertices sent by the VGT:

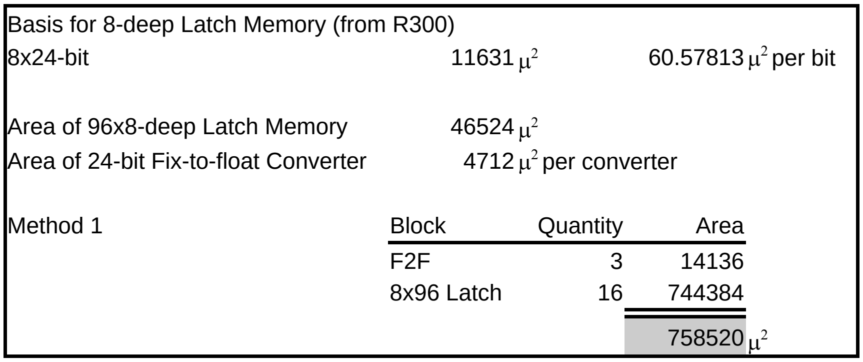
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 || 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 || 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 || 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63

The sequencer will re-arrange them in this fashion:

0 1 2 3 16 17 18 19 32 33 34 35 48 49 50 51 || 4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55 || 8 9 10 11 24 25 26 27 40 41 42 43 56 57 58 59 || 12 13 14 15 28 29 30 31 44 45 46 47 60 61 62 63

The || markers show the SP divisions. In the event a shader pipe is broken, the SQ is responsible to insert padding to account for the missing pipe. For example, if SP1 is broken, vertices 4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55 will not be sent by the VGT to the SQ **AND** the SQ is responsible to "jump" over these vertices in order for no valid vertices to be sent to an invalid SP.

The most straightforward, *non-compressed* interface method would be to convert, in the VGT, the data to 32-bit floating point prior to transmission to the VSISRs. In this scenario, the data would be transmitted to (and stored in) the VSISRs in full 32-bit floating point. This method requires three 24-bit fixed-to-float converters in the VGT. Unfortunately, it also requires and additional 3,072 bits of storage across the VSISRs. This interface is illustrated in Figure 9. The area of the fixed-to-float converters and the VSISRs for this method is roughly estimated as 0.759sqmm using the R300 process. The gate count estimate is shown in Figure 8.



Basis for 8-deep Latch Memory (from R300)

8x24-bit 11631 μ^2 60.57813 μ^2 per bit

Area of 96x8-deep Latch Memory 46524 μ^2

Area of 24-bit Fix-to-float Converter 4712 μ^2 per converter

Method 1

Block	Quantity	Area
F2F	3	14136
8x96 Latch	16	744384
		758520 μ^2

Figure 8:Area Estimate for VGT to Shader Interface

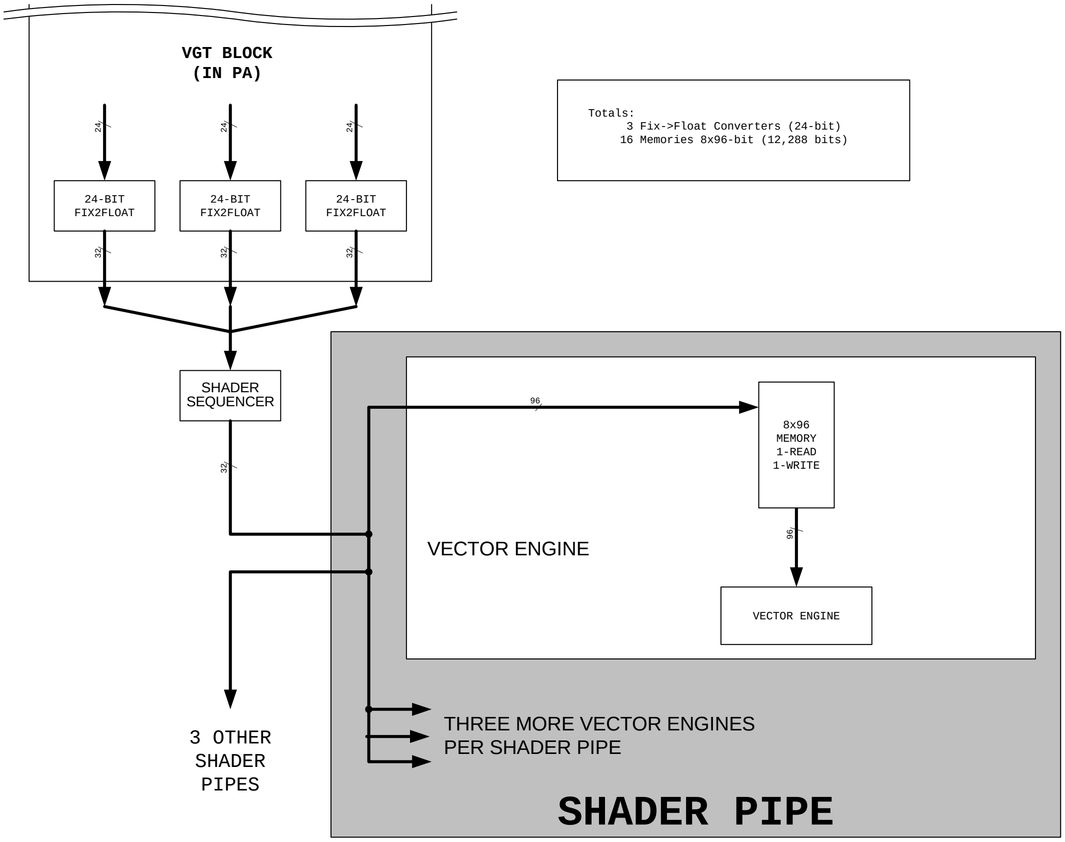


Figure 9:VGT to Shader Interface

17. The parameter cache

The parameter cache is where the vertex shaders export their data. It consists of 16 128x128 memories (1R/1W). The reuse engine will make it so that all vertexes of a given primitive will hit different memories. The allocation method for these memories is a simple round robin. The parameter cache pointers are mapped in the following way: 4MSBs are the memory number and the 7 LSBs are the address within this memory.

MEMORY NUMBER 4 bits	ADDRESS 7 bits

The PA generates the parameter cache addresses as the positions come from the SQ. All it needs to do is keep a Current_Location pointer (7 bits only) and as the positions comes increment the memory number. When the memory number field wraps around, the PA increments the Current_Location by VS_EXPORT_COUNT (a snooped register from the SQ). As an example, say the memories are all empty to begin with and the vertex shader is exporting 8 parameters per vertex (VS_EXPORT_COUNT = 8). The first position received is going to have the PC address 00000000000 the second one 00010000000, third one 00100000000 and so on up to 11110000000. Then the next position received (the 17th) is going to have the address 00000001000, the 18th 00010001000, the 19th 00100001000 and so on. The Current_location is NEVER reset BUT on chip resets. The only thing to be careful about is that if the SX doesn't send you a full group of positions (<64) then you need to fill the address space so that the next group starts correctly aligned (for example if you receive only 33 positions then you need to add 2*VS_EXPORT_COUNT to Current_Location and reset the memory count to 0 before the next vector begins).

17.1 Export restrictions

17.1.1 *Pixel exports:*

Pixels can export 1,2,3 or 4 color buffers to the SX(+z). The exports will be done in order. The exports will always be ordered to the SX.

17.1.2 *Vertex exports:*

Position or parameter caches can be exported in any order in the shader program. It is always better to export posistion as soon as possible. Position has to be exported in a single export block (no texture instructions can be placed between the exports). Parameter cache exports can be done in any order with texture instructions interleaved. The exports will always be allocated in order to the SX.

17.1.3 *Pass thru exports:*

Pass thru exports have to be done in groups of the form:

```
Alloc 1 thru 5 (max export offset + 1, for example if using EM4 alloc size 5)
Execute ALU(ADDR) ALU(DATA) ALU(DATA) ALU(DATA)…
```

When exporting to more than EM0, one MUST write to EM4 also (the write may be predicated if you don’t need the export). This is used to initialize the buffers in the SX.

There cannot be any serialize bits set OR texture Reads between the EA and the last EM.

Memory exports will be surfaced using a macro extension; here is what needs to happen inside the macro:

The macro needs to create a special constant of the form:

Stream ID constant:
.x = Integer that holds BaseAddressInBytes/4 in bits (29:0). Bits 31:30 should be 0b01.
.y = 2**23
.z = Integer that holds register field data. Note that this data must be organized so that it always represents a 'valid' floating point number, with the relevant bits in (23 - 0); One way of doing this would be to take the 23 bits and add 2**23.
.w = max index value + 2**23

Output to EXaddress:

.x = Base of array (in low 30 bits)/4

.y	= Index value (in low 23 bits)
.z	= Register Field data (in low 23 bits)
.w	= Max Index value (in low 23 bits)

Also Assume that C0:

.x	= 0.0
.y	= 1.0

The Macro expansion would be as follows:

```
MULADD      EA = Rindex.xxxx,C0.xyxx,CstreamID;
MOV         EMx (x = 0 thru 4) = Rdata;
```

The SX will check for invalid writes and **mask out the data** so it won't be written to memory. Invalid writes are:

1) Index value >= Max Index value
2) bit 31 != 0 (negative index)
3) bits [30:23] != 23 + IEEE_EXP_BIAS (127) (meaning the index was too big to be represented using 23 bits)

They cannot have texture instructions interleaved in the export block. These exports **are not guaranteed to be ordered**.

Also, when doing a pass thru export**,** the shader must still do either a position and PC export (if Vertex) or a color export (if Pixel). The pass thru export can occur anywhere in any shader program and thus can be used to debug. There can be any number of pass thru export blocks throughout the pixel or vertex shader or both.

17.2 Arbitration restrictions

Here are the Sequencer arbitration restrictions:

1) Cannot execute a serialized thread if the corresponding texture pending bit and VC pending is set
2) Cannot allocate position if any older thread has not allocated position
3) Cannot execute a texture clause if texture reads are pending
4) Cannot execute a VC clause if VC reads are pending
5) Cannot execute last if texture pending (even if not serial)
6) Cannot allocate if not last for color exports.
7) Cannot allocate if not last for PC exports.

18. Export Types

The export type (or the location where the data should be put) is specified using the destination address field in the ALU instruction. Here is a list of all possible export modes:

18.1 Vertex Shading

0:15	- 16 parameter cache
16:31	- Empty (Reserved?)
32	- Export Address
33:37	- 5 vertex exports to the frame buffer and index
38:46	- Empty
47	- Debug Address
48:52	- 5 debug export (interpret as normal memory export)
53:59	- Empty
60	- export addressing mode
61	- Empty
62	- position
63	- sprite size export that goes with position export

(X= point size, Y= edge flag is bit 0, Z= VtxKill is bitwise OR of bits 30:0. Any bit other than sign means VtxKill.)

18.2 Pixel Shading

0	- Color for buffer 0 (primary)
1	- Color for buffer 1
2	- Color for buffer 2
3	- Color for buffer 3
4:15	- Empty
16	- Buffer 0 Color/Fog (primary)
17	- Buffer 1 Color/Fog
18	- Buffer 2 Color/Fog
19	- Buffer 3 Color/Fog
20:31	- Empty
32	- Export Address
33:37	- 5 exports for multipass pixel shaders.
38:46	- Empty
47	- Debug Address
48:52	- 5 debug exports (interpret as normal memory export)
60	- export addressing mode
61	- Z for primary buffer (Z exported to 'alpha' component)
62:63	- Empty

19. Special Interpolation modes

19.1 Real time commands

We are unable to use the parameter memory since there is no way for a command stream to write into it. Instead we need to add three 4x128 memories (one for each of three vertices x 4 interpolants). These will be mapped onto the register bus and written by type 0 packets, and output to the the parameter busses (the sequencer and/or PA need to be able to address the realtime parameter memory as well as the regular parameter store. This mode is triggered by the primitive type: REAL TIME. The actual memories are in the in the SX blocks. The parameter data memories are hooked on the RBBM bus and are loaded by the CP using register mapped memory.

19.2 Sprites/ XY screen coordinates/ FB information

XY screen coordinates may be needed in the shader program. This functionality is controlled by the param_gen register (in SQ) in conjunction with the SND_XY register (in SC) and the param_gen_pos. Also it is possible to send the faceness information (for OGL front/back special operations) to the shader using the same control register. Here is a list of all the modes and how they interact together:

The Data is going to be written in the register specified by the param_gen_pos register.

Param_Gen disable, snd_xy disable = No modification
Param_Gen disable, snd_xy enable = No modification
Param_Gen enable, snd_xy disable = Sign(faceness)garbage,(Sign Point)garbage,Sign(Line)s, t
Param_Gen enable, snd_xy enable = Sign(faceness)screenX,(Sign Point)screenY,Sign(Line)s, t

In other words,

The generated vector is (X in RED, Y in GREEN, S in BLUE and T in ALPHA):
X,Y,S,T
These values are always supposed to be positive and any shader use of them should use the ABS function (as their sign bits will now be used for flags).
SignX = BackFacing
SignY = Point Primitive
SignS = Line Primitive
SignT = currently unused as a flag.

If !Point & !Line, then it is a Poly.

I would assume that one implementation which allows for generic texture lookup (using 3D maps) for poly stipple and AA for the driver would be

```
if(Y<0) {
        R = 0.0 (Point)
} else if (S < 0) {
        R = 1.0 (Line)
} else {
        R = 2.0 (Poly)
}
```

19.3 Auto generated counters

In the cases we are dealing with multipass shaders, the sequencer is going to generate a vector count to be able to both use this count to write the 1st pass data to memory and then use the count to retrieve the data on the 2nd pass. The count is always generated in the same way but it is passed to the shader in a slightly different way depending on the shader type (pixel or vertex). This is toggled on and off using the GEN_INDEX_PIX/VTX register. The sequencer is going to keep two counters, one for pixels and one for vertices. Every time a full vector of vertices or pixels is written to the GPRs the counter is incremented. Every time a RST_PIX_COUNT or RST_VTX_COUNT events are received, the corresponding counter is reset. While there is only one count broadcast to the GPRs, the LSB are hardwired to specific values making the index different for all elements in the vector. Since the count must be different for all pixels/vertices and the 4 LSBs (16 positions) are hardwired to the corresponding shader unit the SQ has two choices:

1) Maintain a 19 bit counter that counts the vectors of 64. In this case the phase must be appended to the count before the count is broadcast to the SPs:

Counter (19 bits)	Phase (2 bits)	Hardwired (4 bits)

2) Maintain a 21 bits counter that counts sub-vectors of 16. In this case only the counter is sent to the Sps:

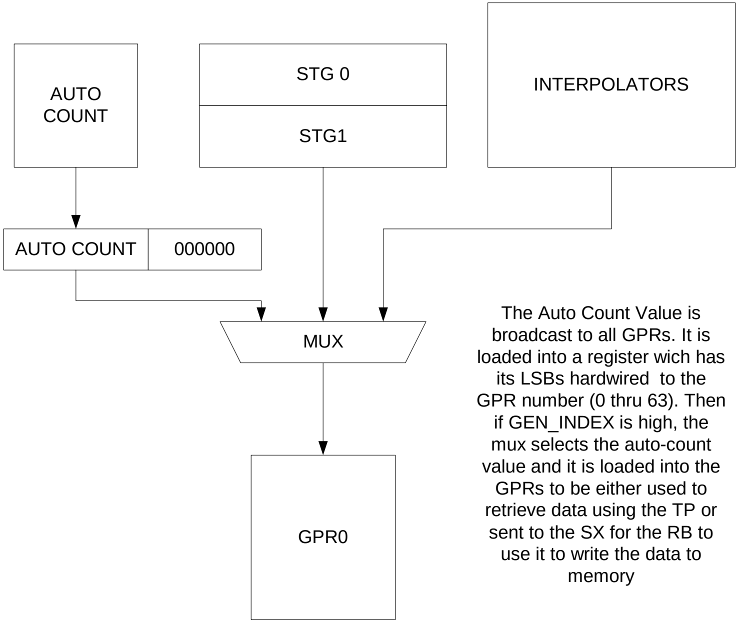
Counter (21 bits)	Hardwired (4 bits)

19.3.1 *Vertex shaders*

In the case of vertex shaders, if GEN_INDEX_VTX is set, the data will be put into the x field of the third register (it means that the compiler must allocate 3 GPRs in all multipass vertex shader modes).

19.3.2 *Pixel shaders*

In the case of pixel shaders, if GEN_INDEX_PIX is set, the data will be put in the x field of the param_gen_pos+1 register.



The Auto Count Value is broadcast to all GPRs. It is loaded into a register wich has its LSBs hardwired to the GPR number (0 thru 63). Then if GEN_INDEX is high, the mux selects the auto-count value and it is loaded into the GPRs to be either used to retrieve data using the TP or sent to the SX for the RB to use it to write the data to memory

Figure 10: GPR input mux Control

20. State management

Every clock, the sequencer will report to the CP the oldest states still in the pipe. These are the states of the programs as they enter the last ALU clause.

20.1 Parameter cache synchronization

In order for the sequencer not to begin a group of pixels before the associated group of vertices has finished, the sequencer will keep a 6 bit count per state (for a total of 8 counters). These counters are initialized to 0 and every time a vertex shader exports its data TO THE PARAMETER CACHE, the corresponding pointer is incremented. When the SC sends a new vector of pixels with the SC_SQ_new_vector bit asserted, the sequencer will first check if the count is greater than 0 before accepting the transmission (it will in fact accept the transmission but then lower its ready to receive). Then the sequencer waits for the count to go to one and decrements it. The sequencer can then issue the group of pixels to the interpolators. Every time the state changes, the new state counter is initialized to 0.

21. XY Address imports

The SC will be able to send the XY addresses to the GPRs. It does so by interleaving the writes of the IJs (to the IJ buffer) with XY writes (to the XY buffer). Then when writing the data to the GPRs, the sequencer is going to interpolate the IJ data or pass the XY data thru a Fix→float converter and expander and write the converted values to the GPRs. The Xys are currently SCREEN SPACE COORDINATES. The values in the XY buffers will wrap. See section 19.2 for details on how to control the interpolation in this mode.

21.1 Vertex indexes imports

In order to import vertex indexes, we have 16 8x96 staging registers. These are loaded one line at a time by the VGT block (96 bits). They are loaded in floating point format and can be transferred in 4 or 8 clocks to the GPRs.

22. Registers

Please see the auto-generated web pages for register definitions.

23. Interfaces

23.1 External Interfaces

Whenever an x is used, it means that the bus is broadcast to all units of the same name. For example, if a bus is named SQ→SPx it means that SQ is going to broadcast the same information to all SP instances.

23.2 SC to SP Interfaces

23.2.1 *SC_SP#*

There is one of these interfaces at front of each of the SP (buffer to stage pixel interpolators). This interface transmits the I,J data for pixel interpolation. For the entire system, two quads per clock are transferred to the 4 SPs, so each of these 4 interfaces transmits one half of a quad per clock. The interface below describes a half of a quad worth of data.
The actual data which is transferred per quad is
Ref Pix I => S4.20 Floating Point I value *4
Ref Pix J => S4.20 Floating Point J value *4

This equates to a total of 200 bits which transferred over 2 clocks
and therefor needs an interface 100 bits wide

Additionally, X,Y data (12-bit unsigned fixed) is conditionally sent across this data bus over the same wires in an additional clock. The X,Y data is sent on the lower 24 bits of the data bus with faceness in the msb.
Transfers across these interfaces are synchronized with the SC_SQ IJ Control Bus transfers.

The data transfer across each of these busses is controlled by a IJ_BUF_INUSE_COUNT in the SC. Each time the SC has sent a pixel vector's worth of data to the SPs, he will increment the IJ_BUF_INUSE_COUNT count. Prior to sending the next pixel vectors data, he will check to make sure the count is less than MAX_BUFER_MINUS_2, if not the SC will stall until the SQ returns a pipelined pulse to decrement the count when he has scheduled a buffer free. Note: We could/may optimize for the case of only sending only IJ to use all the buffers to pre-load more. Currently it is planned for the SP to hold 2 double buffers of I,J data and two buffers of X,Y data, so if either X,Y or Centers and Centroids are on, then the SC can send two Buffers.

In at least the initial version, the SC shall send 16 quads per pixel vector even if the vector is not full. This will increment buffer write address pointers correctly all the time. (We may revisit this for both the SX,SP,SQ and add a EndOfVector signal on all interfaces to quit early. We opted for the simple mode first with a belief that only the end of packet and multiple new vector signals should cause a partial vector and that this would not really be significant performance hit.)

Name	Bits	Description
SC_SP#_data	100	IJ information sent over 2 clocks (or X,Y in 24 LSBs with faceness in upper bit) **Type 0 or 1**, First clock I, second clk J Field ULC URC LLC LRC Bits [63:39] [38:26] [25:13] [12:0] Format SE4M20 SE4M20 SE4M20 SE4M20 **Type 2** Field Face X Y Bits [24] [23:12] [11:0] Format Bit Unsigned Unsigned
SC_SP#_valid	1	Valid
SC_SP#_last_quad_data	1	This bit will be set on the last transfer of data per quad.
SC_SP#_type	2	0 -> Indicates centroids 1 -> Indicates centers 2 -> Indicates X,Y Data and faceness on data bus The SC shall look at state data to determine how many types to send for the

		interpolation process.

The # is included for clarity in the spec and will be replaced with a prefix of u#_ in the verilog module statement for the SC and the SP block will have neither because the instantiation will insert the prefix.

23.2.2 *SC_SQ*

This is the control information sent to the sequencer in order to synchronize and control the interpolation and/or loading data into the GPRs needed to execute a shader program on the sent pixels. This data will be sent over two clocks per transfer with 1 to 16 transfers. Therefore the bus (approx 108 bits) could be folded in half to approx 54 bits.

Name	**Bits**	**Description**
SC_SQ_data	46	Control Data sent to the SQ 1 clk transfers Event – valid data consist of event_id and state_id. Instruct SQ to post an event vector to send state id and event_id through request fifo and onto the reservation stations making sure state id and/or event_id gets back to the CP. Events only follow end of packets so no pixel vectors will be in progress. Empty Quad Mask – Transfer Control data consisting of pc_dealloc or new_vector. Receipt of this is to transfer pc_dealloc or new_vector without any valid quad data. New vector will always be posted to request fifo and pc_dealloc will be attached to any pixel vector outstanding or posted in request fifo if no valid quad outstanding. 2 clk transfers Quad Data Valid – Sending quad data with or without new_vector or pc_dealloc. New vector will be posted to request fifo with or without a pixel vector and pc_dealloc will be posted with a pixel vector unless none is in progress. In this case the pc_dealloc will be posted in the request queue. Filler quads will be transferred with The Quad mask set but the pixel corresponding pixel mask set to zero.
SC_SQ_valid	1	SC sending valid data, 2nd clk could be all zeroes

SC_SQ_data – first clock and second clock transfers are shown in the table below.

Name	**BitField**	**Bits**	**Description**
1st Clock Transfer			
SC_SQ_event	0	1	This transfer is a 1 clock event vector Force quad_mask = new_vector=pc_dealloc=0
SC_SQ_event_id	[5:1]	4	This field identifies the event 0 => denotes an End Of State Event 1

			=> TBD
SC_SQ_state_id	[8:6]	3	State/constant pointer (6*3+3)
SC_SQ_pc_dealloc	[11:9]	3	Deallocation token for the Parameter Cache
SC_SQ_new_vector	12	1	The SQ must wait for Vertex shader done count > 0 and after dispatching the Pixel Vector the SQ will decrement the count.
SC_SQ_quad_mask	[16:13]	4	Quad Write mask left to right SP0 => SP3
SC_SQ_end_of_prim	17	1	End Of the primitive
SC_SQ_pix_mask	[33:18]	16	Valid bits for all pixels SP0=>SP3 (UL,UR,LL,LR)
SC_SQ_provok_vtx	[35:34]	2	Provoking vertex for flat shading
SC_SQ_lod_correct_0	[44:36]	9	LOD correction for quad 0 (SP0) (9 bits per quad)
SC_SQ_lod_correct_1	[53:45]	9	LOD correction for quad 1 (SP1) (9 bits per quad)
2nd Clock Transfer			
SC_SQ_lod_correct_2	[8:0]	9	LOD correction for quad 2 (SP2) (9 bits per quad)
SC_SQ_lod_correct_3	[17:9]	9	LOD correction for quad 3 (SP3) (9 bits per quad)
SC_SQ_pc_ptr0	[28:18]	11	Parameter Cache pointer for vertex 0
SC_SQ_pc_ptr1	[39:29]	11	Parameter Cache pointer for vertex 1
SC_SQ_pc_ptr2	[50:40]	11	Parameter Cache pointer for vertex 2
SC_SQ_prim_type	[53:51]	3	Stippled line and Real time command need to load tex cords from alternate buffer 000: Sprite (point) 001: Line 010: Tri_rect 100: Realtime Sprite (point) 101: Realtime Line 110: Realtime Tri_rect

Name	**Bits**	**Description**
SQ_SC_free_buff	1	Pipelined bit that instructs SC to decrement count of buffers in use.
SQ_SC_dec_cntr_cnt	1	Pipelined bit that instructs SC to decrement count of new vector and/or event sent to prevent SC from overflowing SQ interpolator/Reservation request fifo.

The scan converter will submit a partial vector whenever:

1.) He gets a primitive marked with an end of packet signal.

2.) A current pixel vector is being assembled with at least one or more valid quads and the vector has been marked for deallocate when a primitive marked new_vector arrives. The Scan Converter will submit a partial vector (up to 16quads with zero pixel mask to fill out the vector) prior to submitting the new_vector marker\primitive.

(This will prevent a hang which can be demonstrated when all primitives in a packet three vectors are culled except for a one quad primitive that gets marked pc_dealloc (vertices maximum size). In this case two new_vectors are submitted and processed, but then one valid quad with the pc_dealloc creates a vector and then the new would wait for another vertex vector to be processed, but the one being waited for could never export until the pc_dealloc signal made it through and thus the hang.)

23.2.3 *SQ to SX(SP): Interpolator bus*

Name	Direction	Bits	Description
SQ_SPx_interp_flat_vtx	SQ→SPx	2	Provoking vertex for flat shading
SQ_SPx_interp_flat_gouraud	SQ→SPx	1	Flat or gouraud shading
SQ_SPx_interp_cyl_wrap	SQ→SPx	4	Wich channel needs to be cylindrical wrapped
SQ_SPx_interp_param_gen	SQ→SPx	1	Generate Parameter
SQ_SPx_interp_prim_type	SQ→SPx	2	Bits [1:0] of primitive type sent by SC
SQ_SPx_interp_buff_swap	SQ→SPx	1	Swapp IJ buffers
SQ_SPx_interp_IJ_line	SQ→SPx	2	IJ line number
SQ_SPx_interp_mode	SQ→SPx	1	Center/Centroid sampling
SQ_SXx_pc_ptr0	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_pc_ptr1	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_pc_ptr2	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_rt_sel	SQ→SXx	1	Selects between RT and Normal data (Bit 2 of prim type)
SQ_SX0_pc_wr_en	SQ→SX0	8	Write enable for the PC memories
SQ_SX1_pc_wr_en	SQ→SX1	8	Write enable for the PC memories
SQ_SXx_pc_wr_addr	SQ→SXx	7	Write address for the PCs
SQ_SXx_pc_channel_mask	SQ→SXx	4	Channel mask
SQ_SXx_pc_ptr_valid	SQ→SXx	1	Read pointers are valid.
SQ_SPx_interp_valid	SQ→SPx	1	Interpolation control valid
SQ_SPx_SIMD_engine	SQ→SPx	1	Tells which SIMD engine this data belongs to

23.2.4 *SQ to SP: Staging Register Data*

This is a broadcast bus that sends the VSISR information to the staging registers of the shader pipes.

Name	Direction	Bits	Description
SQ_SPx_vsr_data	SQ→SPx	96	Pointers of indexes or HOS surface information
SQ_SPx_vsr_wrt_addr	SQ→SPx	3	Staging register write address
SQ_SPx_vsr_rd_addr	SQ→SPx	3	Staging register read address
SQ_SP0_ vsr_valid	SQ→SP0	1	Data is valid
SQ_SP1_ vsr_ valid	SQ→SP1	1	Data is valid
SQ_SP2_ vsr_ valid	SQ→SP2	1	Data is valid
SQ_SP3_ vsr_ valid	SQ→SP3	1	Data is valid
SQ_SPx_vsr_read	SQ→SPx	1	Increment the read pointers

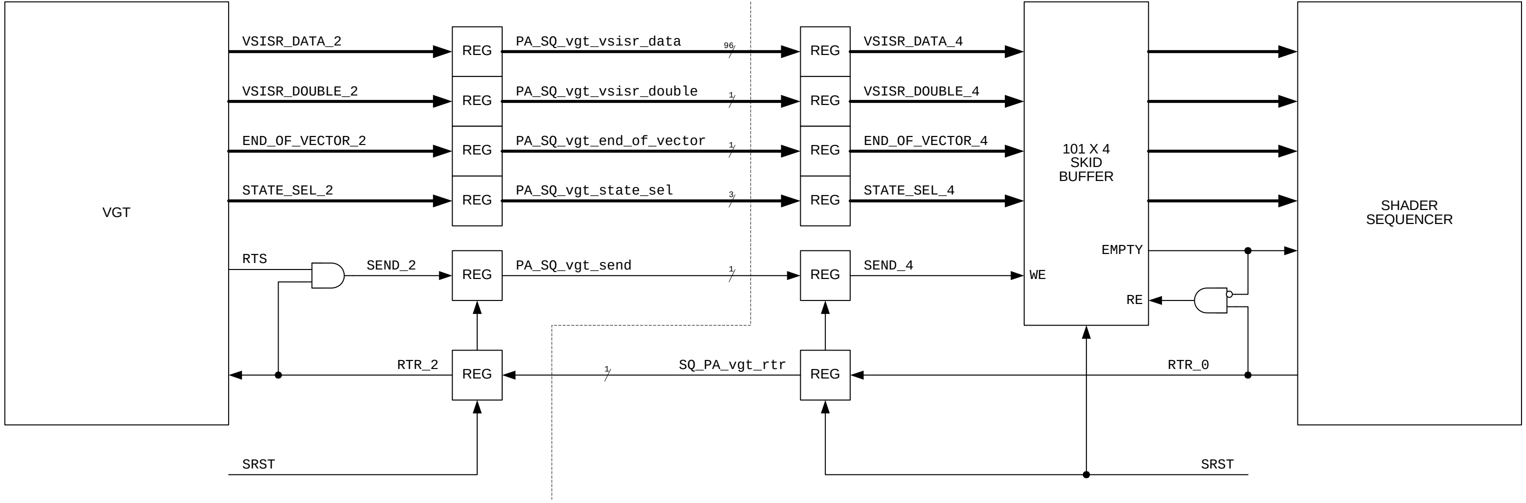
23.2.5 *VGT to SQ : Vertex interface*

23.2.5.1 Interface Signal Table

The area difference between the two methods is not sufficient to warrant complicating the interface or the state requirements of the VSISRs. **Therefore, the POR for this interface is that the VGT will transmit the data to the VSISRs (via the Shader Sequencer) in full, 32-bit floating-point format.** The VGT can transmit up to six 32-bit floating-point values to each VSISR where four or more values require two transmission clocks. The data bus is 96 bits wide. In the case where an event is sent the 5 LSBs of VGT_SQ_vsisr_data contain the eventID.

Name	Bits	Description
VGT_SQ_vsisr_data	96	Pointers of indexes or HOS surface information
VGT_SQ_event	1	VGT is sending an event
VGT_SQ_vsisr_continued	1	0: Normal 96 bits per vert 1: double 192 bits per vert
VGT_SQ_end_of_vtx_vect	1	Indicates the last VSISR data set for the current process vector (for double vector data, "end_of_vector" is set on the first vector)
VGT_SQ_indx_valid	1	Vsisr data is valid
VGT_SQ_state	3	Render State (6*3+3 for constants). This signal is guaranteed to be correct when “VGT_SQ_vgt_end_of_vector” is high.
VGT_SQ_send	1	Data on the VGT_SQ is valid receive (see write-up for standard R400 SEND/RTR interface handshaking)
SQ_VGT_rtr	1	Ready to receive (see write-up for standard R400 SEND/RTR interface handshaking)

23.2.5.2 Interface Diagrams



VGT
VSISR_DATA_2
VSISR_DOUBLE_2
END_OF_VECTOR_2
STATE_SEL_2
RTS
SEND_2
RTR_2
SRST
REG
PA_SQ_vgt_vsisr_data
96
PA_SQ_vgt_vsisr_double
1
PA_SQ_vgt_end_of_vector
1
PA_SQ_vgt_state_sel
3
PA_SQ_vgt_send
1
SQ_PA_vgt_rtr
1
VSISR_DATA_4
VSISR_DOUBLE_4
END_OF_VECTOR_4
STATE_SEL_4
SEND_4
101 X 4
SKID
BUFFER
EMPTY
WE
RE
RTR_0
SRST
SHADER
SEQUENCER

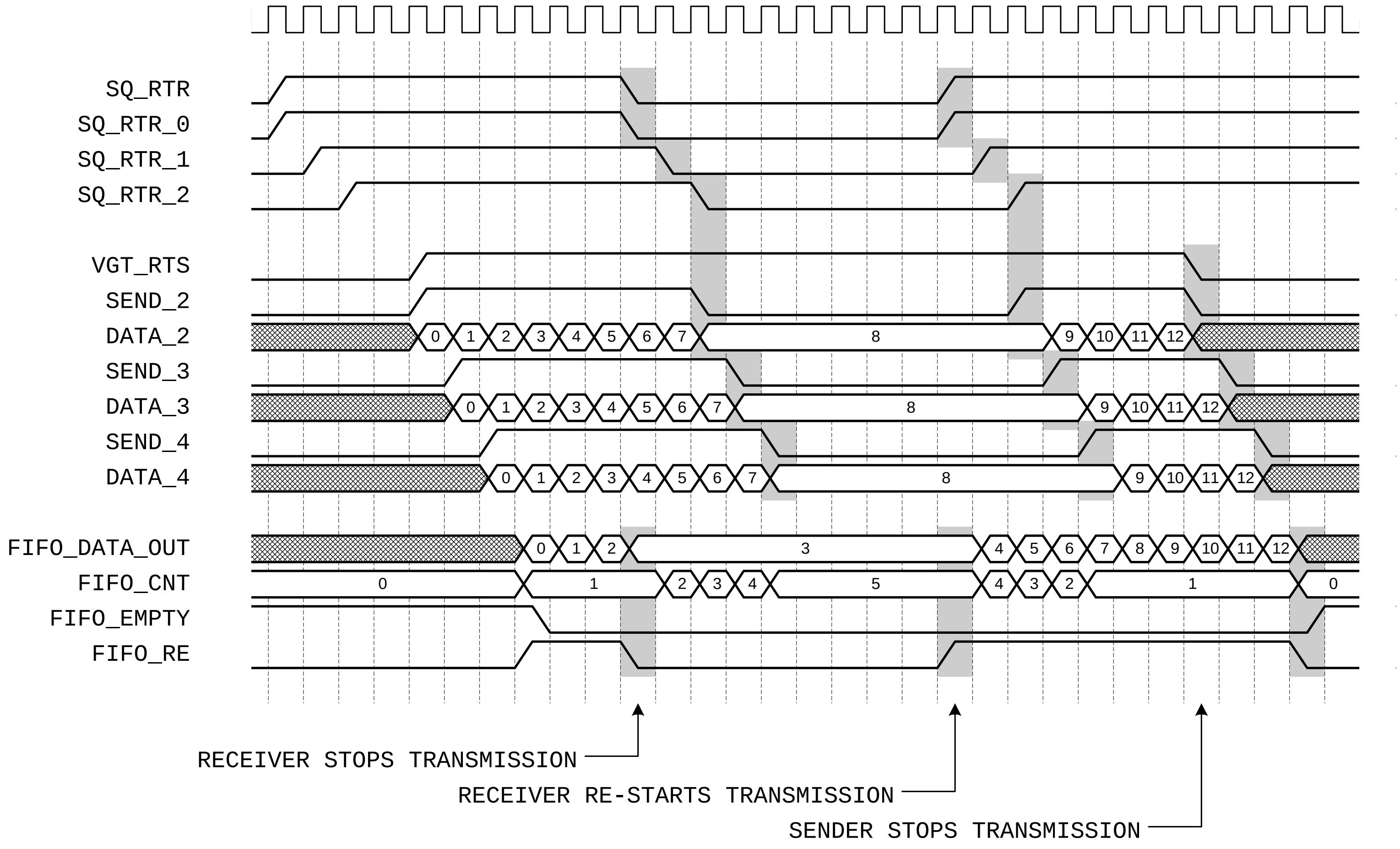


Figure 1. Detailed Logical Diagram for PA_SQ_vgt Interface.

23.2.6 *SQ to SX: Control bus*

Name	Direction	Bits	Description
SQ_SXx_exp_type	SQ→SXx	2	00: Pixel without z (1 to 4 buffers) 01: Pixel with z (1 to 4 buffers) 10: Position (1 or 2 results) 11: Pass thru (1 to 5 results aligned)
SQ_SXx_exp_number	SQ→SXx	2	Number of locations needed in the export buffer (encoding depends on the type see bellow).
SQ_SXx_exp_alu_id	SQ→SXx	4	ALU ID. Revolving ID 0 thru 15. Memory exports have to increment this count by 4 or 8 depending on the size requested. Other type of exports increment the ID by 1.
SQ_SXx_exp_valid	SQ→SXx	1	Valid bit
SQ_SXx_exp_state	SQ→SXx	3	State Context
SQ_SXx_free_done	SQ→SXx	1	Pulse that indicates that the previous export is finished **from the point of view of the SP. This does not necessarily mean that the data has been transferred to RB or PA, or that the space in export buffer for that particular vector thread has been freed up.**
SQ_SXx_free_alu_id	SQ→SXx	4	ALU ID that was used at allocate time.

Depending on the type the number of export location changes:

- Type 00 : Pixels without Z
  - 00 = 1 buffer
  - 01 = 2 buffers
  - 10 = 3 buffers
  - 11 = 4 buffer
- Type 01: Pixels with Z
  - 00 = 2 Buffers (color + Z)
  - 01 = 3 buffers (2 color + Z)
  - 10 = 4 buffers (3 color + Z)
  - 11 = 5 buffers (4 color + Z)
- Type 10 : Position export
  - 00 = 1 position
  - 01 = 2 positions
  - 1X = Undefined
- Type 11: Pass Thru
  - 00 = 4 buffers
  - 01 = 8 buffers
  - 10 = Undefined
  - 11 = Undefined

Below the thick black line is the end of transfer packet that tells the SX that a given export is finished. The report packet **will always arrive either before or at the same time than the next export to the same ALU id**.

23.2.7 *SX to SQ : Output file control*

Name	Direction	Bits	Description
SXx_SQ_pix_free_count0	SXx→SQ	6	How many slots where just freed in the SX for bank0
SXx_SQ_pix_count0_valid	SXx→SQ	1	Free_count0 is valid
SXx_SQ_pix_free_count1	SXx→SQ	6	How many slots where just freed in the SX for bank1
SXx_SQ_pix_count1_valid	SXx→SQ	1	Free_count1 is valid
SXx_SQ_pos_free_count0	SXx→SQ	4	How many slots where just freed in the SX for bank0
SXx_SQ_pos_count0_valid	SXx→SQ	1	Free_count0 is valid
SXx_SQ_pos_free_count1	SXx→SQ	4	How many slots where just freed in the SX for bank1

SXx_SQ_pos_count1_valid	SXx→SQ	1	Free_count1 is valid
SXx_SQ_mem_export_free	SXx→SQ	1	Freed a memory export slot

23.2.8 *SQ to TP: Control bus*

Once every clock, the fetch unit sends to the sequencer on which RS line it is now working and if the data in the GPRs is ready or not. This way the sequencer can update the fetch valid bits flags for the reservation station. The sequencer also provides the instruction and constants for the fetch to execute and the address in the register file where to write the fetch return data.

Name	Direction	Bits	Description
TPx_SQ_data_rdy	TPx→ SQ	1	Data ready
TPx_SQ_rs_line_num	TPx→ SQ	6	Line number in the Reservation station
TPx_SQ_type	TPx→ SQ	1	Type of data sent (0:PIXEL, 1:VERTEX)
SQ_TPx_send	SQ→TPx	1	Sending valid data
SQ_TPx_const	SQ→TPx	48	Fetch state sent over 4 clocks (192 bits total)
SQ_TPx_instr	SQ→TPx	24	Fetch instruction sent over 4 clocks
SQ_TPx_end_of_group	SQ→TPx	1	Last instruction of the group
SQ_TPx_Type	SQ→TPx	1	Type of data sent (0:PIXEL, 1:VERTEX)
SQ_TPx_gpr_phase	SQ→TPx	2	Write phase signal
SQ_TP0_lod_correct	SQ→TP0	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP0_pix_mask	SQ→TP0	4	Pixel mask 1 bit per pixel
SQ_TP1_lod_correct	SQ→TP1	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP1_pix_mask	SQ→TP1	4	Pixel mask 1 bit per pixel
SQ_TP2_lod_correct	SQ→TP2	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP2_pix_mask	SQ→TP2	4	Pixel mask 1 bit per pixel
SQ_TP3_lod_correct	SQ→TP3	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP3_pix_mask	SQ→TP3	4	Pixel mask 1 bit per pixel
SQ_TPx_rs_line_num	SQ→TPx	6	Line number in the Reservation station
SQ_TPx_write_gpr_index	SQ->TPx	7	Index into Register file for write of returned Fetch Data
SQ_TPx_ctx_id	SQ→TPx	3	The state context ID (needed for multisample resolves)
SQ_TPx_SIMD	SQ->TPx	1	Tells the TP from which SIMD the data is coming from.

23.2.9 *SQ to VC: Control bus*

Once every clock, the VC unit sends to the sequencer on which RS line it is now working and if the data in the GPRs is ready or not. This way the sequencer can update the fetch valid bits flags for the reservation station. The sequencer also provides the instruction and constants for the fetch to execute and the address in the register file where to write the fetch return data.

Name	Direction	Bits	Description
VCx_SQ_data_rdy	VCx→ SQ	1	Data ready
VCx_SQ_rs_line_num	VCx→ SQ	6	Line number in the Reservation station
VCx_SQ_type	VCx→ SQ	1	Type of data sent (0:PIXEL, 1:VERTEX)
SQ_VCx_send	SQ→VCx	1	Sending valid data
SQ_VCx_const	SQ→VCx	48	Fetch state sent over 4 clocks (192 bits total)
SQ_VCx_instr	SQ→VCx	24	Fetch instruction sent over 4 clocks
SQ_VCx_end_of_group	SQ→VCx	1	Last instruction of the group
SQ_VCx_Type	SQ→VCx	1	Type of data sent (0:PIXEL, 1:VERTEX)
SQ_VCx_gpr_phase	SQ→VCx	2	Write phase signal
SQ_VC0_pix_mask	SQ→VC0	4	Pixel mask 1 bit per pixel
SQ_VC1_pix_mask	SQ→VC1	4	Pixel mask 1 bit per pixel
SQ_VC2_pix_mask	SQ→VC2	4	Pixel mask 1 bit per pixel
SQ_VC3_pix_mask	SQ→VC3	4	Pixel mask 1 bit per pixel
SQ_VCx_rs_line_num	SQ→VCx	6	Line number in the Reservation station

SQ_VCx_write_gpr_index	SQ->VCx	7	Index into Register file for write of returned Fetch Data
SQ_VCx_SIMD	SQ->VCx	1	Tells the VC from which SIMD the data is coming from.

23.2.10 *TP to SQ: Texture stall*

The TP sends this signal to the SQ and the SPs when its input buffer is full. Stall needs to be aligned with the Instruction start.

Name	Direction	Bits	Description
TP_SQ_fetch_stall	TP→ SQ	1	Do not send more texture request if asserted

23.2.11 *VC to SQ: Vertex Cache stall*

The VCsends this signal to the SQ and the SPs when its input buffer is full. Stall needs to be aligned with the Instruction start.

Name	Direction	Bits	Description
VC_SQ_fetch_stall	VC→ SQ	1	Do not send more vertex cache request if asserted

23.2.12 *SQ to SP: GPR and auto counter*

Name	Direction	Bits	Description
SQ_SPx_simd0_gpr_wr_addr	SQ→SPx	7	Write address
SQ_SPx_simd0_gpr_rd_addr	SQ→SPx	7	Read address
SQ_SPx_simd0_gpr_rd_en	SQ→SPx	1	Read Enable
SQ_SP0_simd0_gpr_pspv_wr_en	SQ→SP0 (SP4)	4	Write Enable for the GPRs of SP0 for PS and PV
SQ_SP1_simd0_gpr_pspv_wr_en	SQ→SP1 (SP5)	4	Write Enable for the GPRs of SP1 for PS and PV
SQ_SP2_simd0_gpr_pspv_wr_en	SQ→SP2 (SP6)	4	Write Enable for the GPRs of SP2 for PS and PV
SQ_SP3_simd0_gpr_pspv_wr_en	SQ→SP3 (SP7)	4	Write Enable for the GPRs of SP3 for PS and PV
SQ_SP0_simd0_gpr_int_wr_en	SQ→SP0	1	Write Enable for the GPRs of SP0 for Inputs (interp/vtx)
SQ_SP1_simd0_gpr_int_wr_en	SQ→SP1	1	Write Enable for the GPRs of SP1 for Inputs (interp/vtx)
SQ_SP2_simd0_gpr_int_wr_en	SQ→SP2	1	Write Enable for the GPRs of SP2 for Inputs (interp/vtx)
SQ_SP3_simd0_gpr_int_wr_en	SQ→SP3	1	Write Enable for the GPRs of SP3 for Inputs (interp/vtx)
SQ_SPx_gpr_phase	SQ→SPx	2	The phase mux (arbitrates between inputs, ALU SRC reads and writes)
SQ_SPx_simd0_channel_mask	SQ→SPx	4	The channel mask for SIMD0
SQ_SPx_gpr_input_sel	SQ→SPx	2	When the phase mux selects the inputs this tells from which source to read from: Interpolated data, VTX0, VTX1, autogen counter.
SQ_SPx_auto_count	SQ→SPx	21	Auto count generated by the SQ, common for all shader pipes
SQ_SPx_simd0_fetch_swizzle	SQ→SPx	6	Swizzle code for the TP request (2 bits per channel ignore W as it is not used). Bits [1..0] X mode select: 0=GPR_X 1=GPR_Y 2=GPR_Z 3=GPR_W Bits [3..2] Y mode select: 0=GPR_X 1=GPR_Y 2=GPR_Z 3=GPR_W Bits [5..4] Z mode select: 0=GPR_X 1=GPR_Y 2=GPR_Z 3=GPR_W
SQ_SPx_simd0_fetch_resource	SQ→SPx	1	Resource in use currently 0: TP 1: VC
SQ_SPx_simd1_gpr_wr_addr	SQ→SPx	7	Write address
SQ_SPx_simd1_gpr_rd_addr	SQ→SPx	7	Read address
SQ_SPx_simd1_gpr_rd_en	SQ→SPx	1	Read Enable
SQ_SP0_simd1_gpr_pspv_wr_en	SQ→SP0 (SP4)	4	Write Enable for the GPRs of SP0 for PS and PV

SQ_SP1_simd1_gpr_pspv_wr_en	SQ→SP1 (SP5)	4	Write Enable for the GPRs of SP1 for PS and PV
SQ_SP2_simd1_gpr_pspv_wr_en	SQ→SP2 (SP6)	4	Write Enable for the GPRs of SP2 for PS and PV
SQ_SP3_simd1_gpr_pspv_wr_en	SQ→SP3 (SP7)	4	Write Enable for the GPRs of SP3 for PS and PV
SQ_SPx__simd1_channel_mask	SQ→SPx	4	The channel mask for SIMD1
SQ_SPx_simd1_fetch_resource	SQ→SPx	1	Resource in use currently 0: TP 1: VC
SQ_SPx_simd1_fetch_swizzle	SQ→SPx	6	Swizzle code for the TP request (2 bits per channel ignore W as it is not used). Bits [1..0] X mode select: 0=GPR_X 1=GPR_Y 2=GPR_Z 3=GPR_W Bits [3..2] Y mode select: 0=GPR_X 1=GPR_Y 2=GPR_Z 3=GPR_W Bits [5..4] Z mode select: 0=GPR_X 1=GPR_Y 2=GPR_Z 3=GPR_W
SQ_SP0_simd1_gpr_int_wr_en	SQ→SP0	1	Write Enable for the GPRs of SP0 for Inputs (interp/vtx)
SQ_SP1_simd1_gpr_int_wr_en	SQ→SP1	1	Write Enable for the GPRs of SP1 for Inputs (interp/vtx)
SQ_SP2_simd1_gpr_int_wr_en	SQ→SP2	1	Write Enable for the GPRs of SP2 for Inputs (interp/vtx)
SQ_SP3_simd1_gpr_int_wr_en	SQ→SP3	1	Write Enable for the GPRs of SP3 for Inputs (interp/vtx)

23.2.13 *SQ to SPx:*

Name	Direction	Bits	Description
SQ_SPx_instr_start	SQ→SPx	1	Instruction start
SQ_SPx_simd0_instruct	SQ→SPx	24	Transferred over 4 cycles 0: SRC A Negate Argument Modifier 0:0 SRC A Abs Argument Modifier 1:1 SRC A Swizzle 9:2 Vector Dst 15:10 Per channel Select 23:16 00: GPR 01: PV 10: PS 11: Constant (if 11 has to be 11 for all channels) -- - 1: SRC B Negate Argument Modifier 0:0 SRC B Abs Argument Modifier 1:1 SRC B Swizzle 9:2 Scalar Dst 15:10 Per channel Select 23:16 00: GPR 01: PV 10: PS 11: Constant (if 11 has to be 11 for all channels) -- - 2: SRC C Negate Argument Modifier 0:0 SRC C Abs Argument Modifier 1:1 SRC C Swizzle 9:2 Unused 15:10 Per channel Select 23:16 00: GPR 01: PV 10: PS 11: Constant (if 11 has to be 11 for all channels) -- - 3: Vector Opcode 4:0 Scalar Opcode 10:5 Vector Clamp 11:11 Scalar Clamp 12:12 Vector Write Mask 16:13 Scalar Write Mask 20:17 Unused 23:21
SQ_SP0_simd0_pred_override	SQ→SP0 (SP4)	4	0: Use per channel RGBA field (enables the per channel logic). 1: Use GPR for PV or PS settings. LET the 11 (constant) go thru unchanged
SQ_SP1_simd0_pred_override	SQ→SP1 (SP5)	4	0: Use per channel RGBA field (enables the per channel logic). 1: Use GPR for PV or PS settings. LET the 11 (constant) go thru unchanged
SQ_SP2_simd0_pred_override	SQ→SP2 (SP6)	4	0: Use per channel RGBA field (enables the per channel logic). 1: Use GPR for PV or PS settings. LET the 11 (constant) go thru unchanged
SQ_SP3_simd0_pred_override	SQ→SP3 (SP7)	4	0: Use per channel RGBA field (enables the per channel logic).

			1: Use GPR for PV or PS settings. LET the 11 (constant) go thru unchanged
SQ_SPx_simd0_stall	SQ→SPx	1	Stall signal

SQ_SPx_simd0_Waterfall	SQ→SPx	2	Use the incoming constant instead of the registered one for the next group of 16. 0 : Normal mode 1: Waterfall on SRCA 2: Waterfall on SRCB 3: Waterfall on SRCC
SQ_SPx_simd1_instruct	SQ→SPx	24	Transferred over 4 cycles 0: SRC A Negate Argument Modifier 0:0 SRC A Abs Argument Modifier 1:1 SRC A Swizzle 9:2 Vector Dst 15:10 Per channel Select 23:16 00: GPR 01: PV 10: PS 11: Constant (if 11 has to be 11 for all channels) -- - 1: SRC B Negate Argument Modifier 0:0 SRC B Abs Argument Modifier 1:1 SRC B Swizzle 9:2 Scalar Dst 15:10 Per channel Select 23:16 00: GPR 01: PV 10: PS 11: Constant (if 11 has to be 11 for all channels) -- - 2: SRC C Negate Argument Modifier 0:0 SRC C Abs Argument Modifier 1:1 SRC C Swizzle 9:2 Unused 15:10 Per channel Select 23:16 00: GPR 01: PV 10: PS 11: Constant (if 11 has to be 11 for all channels) -- - 3: Vector Opcode 4:0 Scalar Opcode 10:5 Vector Clamp 11:11 Scalar Clamp 12:12 Vector Write Mask 16:13 Scalar Write Mask 20:17 Unused 23:21
SQ_SP0_simd1_pred_override	SQ→SP0 (SP4)	4	0: Use per channel RGBA field (enables the per channel logic). 1: Use GPR for PV or PS settings. LET the 11 (constant) go thru unchanged
SQ_SP1_simd1_pred_override	SQ→SP1 (SP5)	4	0: Use per channel RGBA field (enables the per channel logic). 1: Use GPR for PV or PS settings. LET the 11 (constant) go thru unchanged
SQ_SP2_simd1_pred_override	SQ→SP2 (SP6)	4	0: Use per channel RGBA field (enables the per channel logic). 1: Use GPR for PV or PS settings. LET the 11 (constant) go thru unchanged
SQ_SP3_simd1_pred_override	SQ→SP3 (SP7)	4	0: Use per channel RGBA field (enables the per channel

			logic). 1: Use GPR for PV or PS settings. LET the 11 (constant) go thru unchanged
SQ_SPx_simd1_stall	SQ→SPx	1	Stall signal
SQ_SPx_simd1_Waterfall	SQ→SPx	2	Use the incoming constant instead of the registered one for the next group of 16. 0 : Normal mode 1: Waterfall on SRCA 2: Waterfall on SRCB 3: Waterfall on SRCC
SQ_SPx_export_simd_sel	SQ->SPx	1	Which SIMD engine is exporting.

23.2.14 *SQ to SX: write mask interface (must be aligned with the SP data)*

Name	**Direction**	**Bits**	**Description**
SQ_SX0_write_mask	SQ→SP0	8	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock. This is for the data coming of SP0 and SP2.
SQ_SX1_ write_mask	SQ→SP1	8	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock. This is for the data coming of SP1 and SP3.

23.2.15 *SP to SQ: Constant address load/ Predicate Set/Kill set*

Name	Direction	Bits	Description
SP0_SQ_simd0_const_addr	(SP4) SP0→SQ	36	Constant address load 18 bits from SP0 and 18 from SP4.
SP0_SQ_simd0_valid	SP0→SQ	1	Data valid
SP1_SQ_simd0_const_addr	(SP5) SP1→SQ	36	Constant address load
SP1_SQ_simd0_valid	SP1→SQ	1	Data valid
SP2_SQ_simd0_const_addr	(SP6) SP2→SQ	36	Constant address load
SP2_SQ_simd0_valid	SP2→SQ	1	Data valid
SP3_SQ_simd0_const_addr	(SP7) SP3→SQ	36	Constant address load
SP3_SQ_simd0_valid	SP3→SQ	1	Data valid
SP0_SQ_simd0_pred_kill_vector	(SP4) SP0→SQ	4	Data (predicates or kill/mask) 2 bits from SP0 and 2 bits from SP4
SP0_SQ_simd0_pred_kill_valid	SP0->SQ	1	Data valid
SP0_SQ_simd0_pred_kill_type	SP0->SQ	1	0: predicate vector 1: kill/mask vector
SP1_SQ_simd0_pred_kill_vector	(SP5) SP1→SQ	4	Data (predicates or kill/mask)
SP1_SQ_simd0_pred_kill_valid	SP1->SQ	1	Data valid
SP1_SQ_simd0_pred_kill_type	SP1->SQ	1	0: predicate vector 1: kill/mask vector
SP2_SQ_simd0_pred_kill_vector	(SP6) SP2→SQ	4	Data (predicates or kill/mask)
SP2_SQ_simd0_pred_kill_valid	SP2->SQ	1	Data valid
SP2_SQ_simd0_pred_kill_type	SP2->SQ	1	0: predicate vector 1: kill/mask vector
SP3_SQ_simd0_pred_kill_vector	(SP7) SP3→SQ	4	Data (predicates or kill/mask)
SP3_SQ_simd0_pred_kill_valid	SP3->SQ	1	Data valid
SP3_SQ_simd0_pred_kill_type	SP3->SQ	1	0: predicate vector 1: kill/mask vector
SP0_SQ_simd1_const_addr	(SP4) SP0→SQ	36	Constant address load 18 bits from SP0 and 18 from SP4.
SP0_SQ_simd1_valid	SP0→SQ	1	Data valid
SP1_SQ_simd1_const_addr	(SP5) SP1→SQ	36	Constant address load
SP1_SQ_simd1_valid	SP1→SQ	1	Data valid
SP2_SQ_simd1_const_addr	(SP6) SP2→SQ	36	Constant address load
SP2_SQ_simd1_valid	SP2→SQ	1	Data valid
SP3_SQ_simd1_const_addr	(SP7) SP3→SQ	36	Constant address load
SP3_SQ_simd1_valid	SP3→SQ	1	Data valid
SP0_SQ_simd1_pred_kill_vector	(SP4) SP0→SQ	4	Data (predicates or kill/mask) 2 bits from SP0 and 2 bits from SP4
SP0_SQ_simd1_pred_kill_valid	SP0->SQ	1	Data valid
SP0_SQ_simd1_pred_kill_type	SP0->SQ	1	0: predicate vector 1: kill/mask vector
SP1_SQ_simd1_pred_kill_vector	(SP5) SP1→SQ	4	Data (predicates or kill/mask)
SP1_SQ_simd1_pred_kill_valid	SP1->SQ	1	Data valid
SP1_SQ_simd1_pred_kill_type	SP1->SQ	1	0: predicate vector 1: kill/mask vector
SP2_SQ_simd1_pred_kill_vector	(SP6) SP2→SQ	4	Data (predicates or kill/mask)
SP2_SQ_simd1_pred_kill_valid	SP2->SQ	1	Data valid
SP2_SQ_simd1_pred_kill_type	SP2->SQ	1	0: predicate vector 1: kill/mask vector
SP3_SQ_simd1_pred_kill_vector	(SP7) SP3→SQ	4	Data (predicates or kill/mask)
SP3_SQ_simd1_pred_kill_valid	SP3->SQ	1	Data valid
SP3_SQ_simd1_pred_kill_type	SP3->SQ	1	0: predicate vector 1: kill/mask vector

Because of the sharing of the bus none of the MOVA, PREDSET or KILL instructions may be coissued.

23.2.16 *SQ to SPx: constant broadcast*

Name	Direction	Bits	Description

SQ_SPx_simd0_const	SQ→SPx	128	Constant broadcast
SQ_SPx_simd1_const	SQ→SPx	128	Constant broadcast

23.2.17 *SQ to CP: RBBM bus*

Name	Direction	Bits	Description
SQ_RBB_rs	SQ→CP	1	Read Strobe
SQ_RBB_rd	SQ→CP	32	Read Data
SQ_RBBM_nrtrtr	SQ→CP	1	Optional
SQ_RBBM_rtr	SQ→CP	1	Real-Time (Optional)

23.2.18 *CP to SQ: RBBM bus*

Name	Direction	Bits	Description
rbbm_we	CP→SQ	1	Write Enable
rbbm_a	CP→SQ	15	Address -- Upper Extent is TBD (16:2)
rbbm_wd	CP→SQ	32	Data
rbbm_be	CP→SQ	4	Byte Enables
rbbm_re	CP→SQ	1	Read Enable
rbb_rs0	CP→SQ	1	Read Return Strobe 0
rbb_rs1	CP→SQ	1	Read Return Strobe 1
rbb_rd0	CP→SQ	32	Read Data 0
rbb_rd1	CP→SQ	32	Read Data 0
RBBM_SQ_soft_reset	CP→SQ	1	Soft Reset

23.2.19 *SQ to CP: State report*

Name	Direction	Bits	Description
SQ_CP_vs_event	SQ→CP	1	Vertex Shader Event
SQ_CP_vs_eventid	SQ→CP	5	Vertex Shader Event ID
SQ_CP_ps_event	SQ→CP	1	Pixel Shader Event
SQ_CP_ps_eventid	SQ→CP	5	Pixel Shader Event ID

23.3 Example of control flow program execution

We now provide some examples of execution to better illustrate the new design.

Given the program:

Alu 0
Alu 1
Tex 0
Tex 1
Alu 3 Serial
Alu 4
Tex 2
Alu 5
Alu 6 Serial
Tex 3
Alu 7
Alloc Position 1 buffer
Alu 8 Export
Tex 4
Alloc Parameter 3 buffers
Alu 9 Export 0
Tex 5
Alu 10 Serial Export 2
Alu 11 Export 1 End

Would be converted into the following CF instructions:

```
Execute 0 Alu 0 Alu 0 Tex 0 Tex 1 Alu 0 Alu 0 Tex 0 Alu 1 Alu 0 Tex
Execute 0 Alu
Alloc Position 1
Execute 0 Alu 0 Tex
Alloc Param 3
Execute_end 0 Alu 0 Tex 1 Alu 0 Alu
```

And the execution of this program would look like this:

Put thread in Vertex RS:

Control Flow Instruction Pointer (12 bits), (CFP)
Execution Count Marker (3 or 4 bits), (ECM)
Loop Iterators (4x9 bits), (LI)
Call return pointers (4x12 bits), (CRP)
Predicate Bits(4x64 bits), (PB)
Export ID (1 bit), (EXID)
GPR Base Ptr (8 bits), (GPR)
Export Base Ptr (7 bits), (EB)
Context Ptr (3 bits).(CPTR)
LOD correction bits (16x6 bits) (LOD)

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	0	0	0	0	0	0	0	0	0

Valid Thread (VALID)
Texture/ALU engine needed (TYPE)
Texture Reads are outstanding (PENDING)
Waiting on Texture Read to Complete (SERIAL)
Allocation Wait (2 bits) (ALLOC)
    00 – No allocation needed
    01 – Position export allocation needed (ordered export)
    10 – Parameter or pixel export needed (ordered export)
    11 – pass thru (out of order export)
Allocation Size (4 bits) (SIZE)
Position Allocated (POS_ALLOC)
First thread of a new context (FIRST)
Last (1 bit), (LAST)

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	0	0	0	0	0	1	0

Then the thread is picked up for the execution of the first control flow instruction:

```
Execute 0 Alu 0 Alu 0 Tex 0 Tex 1 Alu 0 Alu 0 Tex 0 Alu 1 Alu 0 Tex
```

It executes the first two ALU instructions and goes back to the RS for a resource request change. Here is the state returned to the RS:

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	2	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	0	0	0	0	0	1	0

Then when the texture pipe frees up, the arbiter picks up the thread to issue the texture reads. The thread comes back in this state:

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	4	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	1	0	0	0	1	0

Because of the serial bit the arbiter must wait for the texture to return and clear the PENDING bit before it can pick the thread up. Lets say that the texture reads are complete, then the arbiter picks up the thread and returns it in this state:

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	6	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	0	0	0	0	0	1	0

Again the TP frees up, the arbiter picks up the thread and executes. It returns in this state:

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	7	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	0	0	0	1	0

Now, even if the texture has not returned we can still pick up the thread for ALU execution because the serial bit is not set. The thread will however come back to the RS for the second ALU instruction because it has the serial bit set.

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	8	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	1	0	0	0	1	0

As soon as the TP clears the pending bit the thread is picked up and returns:

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	9	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	0	0	0	0	0	1	0

Picked up by the TP and returns:

```
Execute 0 Alu
```

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
1	0	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	0	0	0	1	0

Picked up by the ALU and returns (lets say the TP has not returned yet):

```
Alloc Position 1
```

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
2	0	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	01	1	0	1	0

If the SX has the place for the export, the SQ is going to allocate and pick up the thread for execution. It returns to the RS in this state:

```
Execute 0 Alu 0 Tex
```

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
3	1	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	1	0	0	0	1	1	0

Now, since the TP has not returned yet, we must wait for it to return because we cannot issue multiple texture requests. The TP returns, clears the PENDING bit and we proceed:

```
Alloc Param 3
```

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
4	0	0	0	0	1	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	10	3	1	1	0

Once again the SQ makes sure the SX has enough room in the Parameter cache before it can pick up this thread.

```
Execute_end 0 Alu 0 Tex 1 Alu 0 Alu
```

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
5	1	0	0	0	1	0	100	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	1	0	0	0	1	1	0

This executes on the TP and then returns:

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
5	2	0	0	0	1	0	100	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	1	0	0	1	1	1

Waits for the TP to return because of the textures reads are pending (and SERIAL in this case). Then executes and does not return to the RS because the LAST bit is set. This is the end of this thread and before dropping it on the floor, the SQ notifies the SX of export completion.

24. Open issues

Need to do some testing on the size of the register file as well as on the register file allocation method (dynamic VS static).

Saving power?

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	9 July, 2003	4 September, 2015	GEN-CXXXXX-REVA	1 of 54

Author: Laurent Lefebvre

Issue To:

Copy No:

R400 Sequencer Specification

SQ

Version 2.11

Overview: This is an architectural specification for the R400 Sequencer block (SEQ). It provides an overview of the required capabilities and expected uses of the block. It also describes the block interfaces, internal sub-blocks, and provides internal state diagrams.

AUTOMATICALLY UPDATED FIELDS:
Document Location: C:\perforce\r400\doc_lib\design\blocks\sq\R400_Sequencer.doc
Current Intranet Search Title: R400 Sequencer Specification

APPROVALS	
Name/Dept	Signature/Date

Remarks:

THIS DOCUMENT CONTAINS CONFIDENTIAL INFORMATION THAT COULD BE SUBSTANTIALLY DETRIMENTAL TO THE INTEREST OF ATI TECHNOLOGIES INC. THROUGH UNAUTHORIZED USE OR DISCLOSURE.

Table Of Contents

1. OVERVIEW 6
1.1 Top Level Block Diagram 8
1.2 Data Flow graph (SP) 9
1.3 Control Graph 10
2. INTERPOLATED DATA BUS 10
3. INSTRUCTION STORE 13
4. SEQUENCER INSTRUCTIONS 13
5. CONSTANT STORES 13
5.1 Memory organizations 13
5.2 Management of the Control Flow Constants 14
5.3 Management of the re-mapping tables 14
5.3.1 R400 Constant management 14
5.3.2 Dirty bits 14
5.3.3 Free List Block 14
5.3.4 De-allocate Block 15
5.3.5 Operation of Incremental model 15
5.4 Constant Store Indexing 16
5.5 Real Time Commands 16
5.6 Constant Waterfalling 16
6. LOOPING AND BRANCHES 17
6.1 The controlling state. 17
6.2 The Control Flow Program 17
6.2.1 Control flow instructions table 18
6.3 Implementation 21
6.4 Data dependant predicate instructions 23
6.5 HW Detection of PV,PS 24
6.6 Register file indexing 24
6.7 Debugging the Shaders 24
6.7.1 Method 1: Debugging registers 24
6.7.2 Method 2: Exporting the values in the GPRs 25
7. PIXEL KILL MASK 25
8. REGISTER FILE ALLOCATION 25
9. FETCH ARBITRATION 26
10. VC ARBITRATION 26
11. ALU ARBITRATION 27
12. HANDLING STALLS 27
12.1 SP stall conditions 27
12.1.1 PS Stalls 27
12.1.2 PV Stalls 27
13. CONTENT OF THE RESERVATION STATION FIFOS 27
14. THE OUTPUT FILE 27
15. IJ FORMAT 27
15.1 Interpolation of constant attributes 28
16. STAGING REGISTERS 28
17. THE PARAMETER CACHE 29

17.1 Export restrictions 30
17.1.1 Pixel exports: 30
17.1.2 Vertex exports: 30
17.1.3 Pass thru exports: 30
17.2 Arbitration restrictions 31
18. EXPORT TYPES 31
18.1 Vertex Shading 31
18.2 Pixel Shading 32
19. SPECIAL INTERPOLATION MODES 32
19.1 Real time commands 32
19.2 Sprites/ XY screen coordinates/ FB information 32
19.3 Auto generated counters 33
19.3.1 Vertex shaders 33
19.3.2 Pixel shaders 34
20. STATE MANAGEMENT 34
20.1 Parameter cache synchronization 34
21. XY ADDRESS IMPORTS 34
21.1 Vertex indexes imports 34
22. REGISTERS 35
23. INTERFACES 35
23.1 External Interfaces 35
23.2 SC to SP Interfaces 35
23.2.1 SC_SP# 35
23.2.2 SC_SQ 36
23.2.3 SQ to SX(SP): Interpolator bus 38
23.2.4 SQ to SP: Staging Register Data 38
23.2.5 VGT to SQ : Vertex interface 38
23.2.6 SQ to SX: Control bus 42
23.2.7 SX to SQ : Output file control 42
23.2.8 SQ to TP: Control bus 43
23.2.9 SQ to VC: Control bus 43
23.2.10 TP to SQ: Texture stall 44
23.2.11 VC to SQ: Vertex Cache stall 44
23.2.12 SQ to SP: GPR and auto counter 44
23.2.13 SQ to SPx: 46
23.2.14 SQ to SX: write mask interface (must be aligned with the SP data) 48
23.2.15 SP to SQ: Constant address load/ Predicate Set/Kill set 49
23.2.16 SQ to SPx: constant broadcast 49
23.2.17 SQ to CP: RBBM bus 50
23.2.18 CP to SQ: RBBM bus 50
23.2.19 SQ to CP: State report 50
23.3 Example of control flow program execution 50
24. OPEN ISSUES 54

Revision Changes:

Rev 0.1 (Laurent Lefebvre) Date: May 7, 2001	First draft.
Rev 0.2 (Laurent Lefebvre) Date : July 9, 2001	Changed the interfaces to reflect the changes in the SP. Added some details in the arbitration section.
Rev 0.3 (Laurent Lefebvre) Date : August 6, 2001	Reviewed the Sequencer spec after the meeting on August 3, 2001.
Rev 0.4 (Laurent Lefebvre) Date : August 24, 2001	Added the dynamic allocation method for register file and an example (written in part by Vic) of the flow of pixels/vertices in the sequencer.
Rev 0.5 (Laurent Lefebvre) Date : September 7, 2001	Added timing diagrams (Vic)
Rev 0.6 (Laurent Lefebvre) Date : September 24, 2001	Changed the spec to reflect the new R400 architecture. Added interfaces.
Rev 0.7 (Laurent Lefebvre) Date : October 5, 2001	Added constant store management, instruction store management, control flow management and data dependant predication.
Rev 0.8 (Laurent Lefebvre) Date : October 8, 2001	Changed the control flow method to be more flexible. Also updated the external interfaces.
Rev 0.9 (Laurent Lefebvre) Date : October 17, 2001	Incorporated changes made in the 10/18/01 control flow meeting. Added a NOP instruction, removed the conditional_execute_or_jump. Added debug registers.
Rev 1.0 (Laurent Lefebvre) Date : October 19, 2001	Refined interfaces to RB. Added state registers.
Rev 1.1 (Laurent Lefebvre) Date : October 26, 2001	Added SEQ→SP0 interfaces. Changed delta precision. Changed VGT→SP0 interface. Debug Methods added.
Rev 1.2 (Laurent Lefebvre) Date : November 16, 2001	Interfaces greatly refined. Cleaned up the spec.
Rev 1.3 (Laurent Lefebvre) Date : November 26, 2001	Added the different interpolation modes.
Rev 1.4 (Laurent Lefebvre) Date : December 6, 2001	Added the auto incrementing counters. Changed the VGT→SQ interface. Added content on constant management. Updated GPRs.
Rev 1.5 (Laurent Lefebvre) Date : December 11, 2001	Removed from the spec all interfaces that weren't directly tied to the SQ. Added explanations on constant management. Added PA→SQ synchronization fields and explanation.
Rev 1.6 (Laurent Lefebvre) Date : January 7, 2002	Added more details on the staging register. Added detail about the parameter caches. Changed the call instruction to a Conditionnal_call instruction. Added details on constant management and updated the diagram.
Rev 1.7 (Laurent Lefebvre) Date : February 4, 2002	Added Real Time parameter control in the SX interface. Updated the control flow section.
Rev 1.8 (Laurent Lefebvre) Date : March 4, 2002	New interfaces to the SX block. Added the end of clause modifier, removed the end of clause instructions.
Rev 1.9 (Laurent Lefebvre) Date : March 18, 2002	Rearangement of the CF instruction bits in order to ensure byte alignement.
Rev 1.10 (Laurent Lefebvre) Date : March 25, 2002	Updated the interfaces and added a section on exporting rules.
Rev 1.11 (Laurent Lefebvre) Date : April 19, 2002	Added CP state report interface. Last version of the spec with the old control flow scheme
Rev 2.0 (Laurent Lefebvre) Date : April 19, 2002	New control flow scheme

Rev 2.01 (Laurent Lefebvre) Date : May 2, 2002	Changed slightly the control flow instructions to allow force jumps and calls.
Rev 2.02 (Laurent Lefebvre) Date : May 13, 2002	Updated the Opcodes. Added type field to the constant/pred interface. Added Last field to the SQ→SP instruction load interface.
Rev 2.03 (Laurent Lefebvre) Date : July 15, 2002	SP interface updated to include predication optimizations. Added the predicate no stall instructions,
Rev 2.04 (Laurent Lefebvre) Date :August 2, 2002	Documented the new parameter generation scheme for XY coordinates points and lines STs.
Rev 2.05 (Laurent Lefebvre) Date : September 10, 2002	Some interface changes and an architectural change to the auto-counter scheme.
Rev 2.06 (Laurent Lefebvre) Date : October 11, 2002	Widened the event interface to 5 bits. Some other little typos corrected.
Rev 2.07 (Laurent Lefebvre) Date : October 14, 2002	Loops, jumps and calls are now using a 13 bit address which allows to jump and call and loop around any control flow addresses (does not requires to be even anymore).
Rev 2.08 (Laurent Lefebvre) Date : October 16, 2002	Clarification updates after discussion with Clay.
Rev 2.09 (Laurent Lefebvre) Date : January 7, 2003	Corrected the SQ→SP staging register interface.
Rev 2.10 (Laurent Lefebvre) Date : April 8, 2003	Adding R500 modifications
Rev 2.11 (Laurent Lefebvre) Date : May 1, 2003	Adding SQ->SP updated interfaces

1. Overview

The sequencer chooses four ALU threads (two from each bank), a vertex cache and a fetch thread to execute, and executes all of the instructions in a block before looking for a new clause of the same type. Two ALU threads are executed interleaved to hide the ALU latency. The arbitrator will give priority to older threads. There are two separate reservation stations, one for pixel vectors and one for vertices vectors. This way a pixel can pass a vertex and a vertex can pass a pixel.

There are also 2 separate ALU banks from which the SQ picks the ALU threads to be executed in parallel.

To support the shader pipe the sequencer also contains the shader instruction store, constant store, control flow constants and texture state. The height shader pipes also execute the same two instructions thus there is only one sequencer for the whole chip but it issues 2 instructions every four clocks.

The sequencer first arbitrates between vectors of 64 vertices that arrive directly from primitive assembly and vectors of 16 quads (64 pixels) that are generated in the scan converter.

The vertex or pixel program specifies how many GPRs it needs to execute. The sequencer will not start the next vector until the needed space is available in the GPRs.

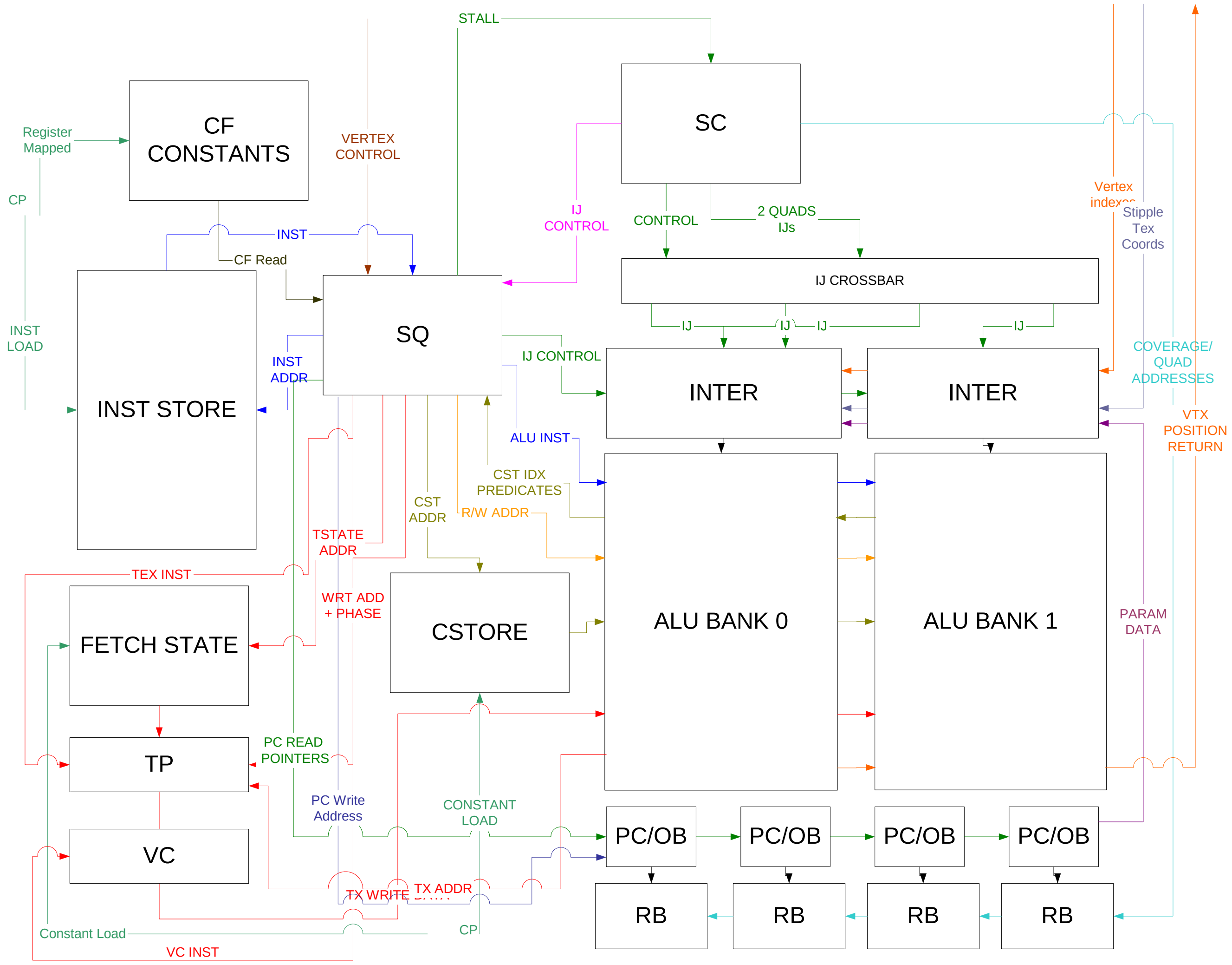


Figure 1: General Sequencer overview

1.1 Top Level Block Diagram

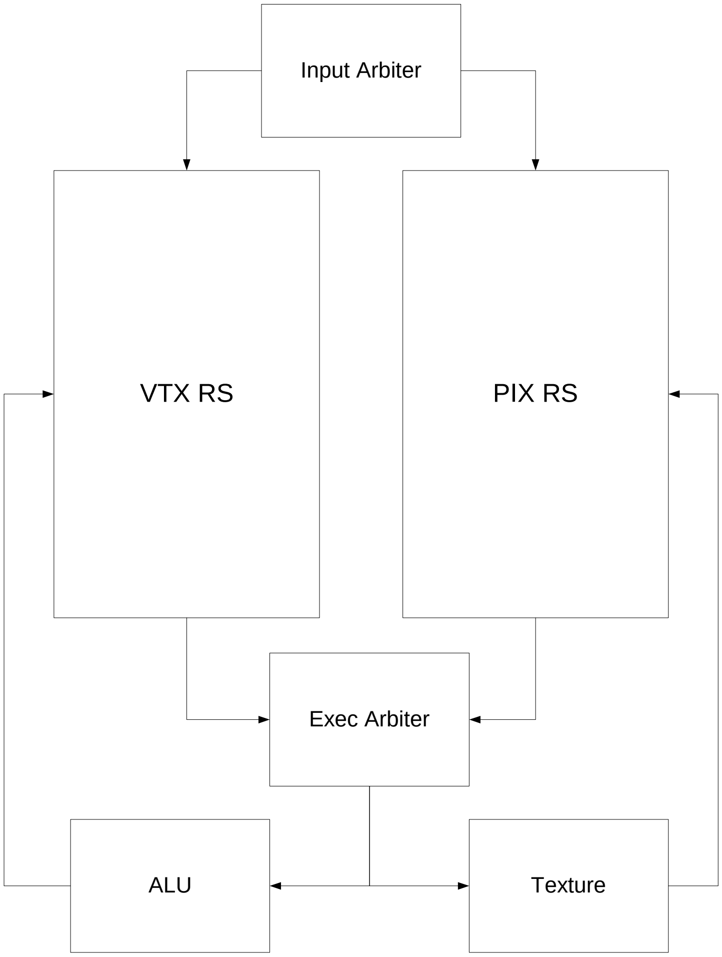
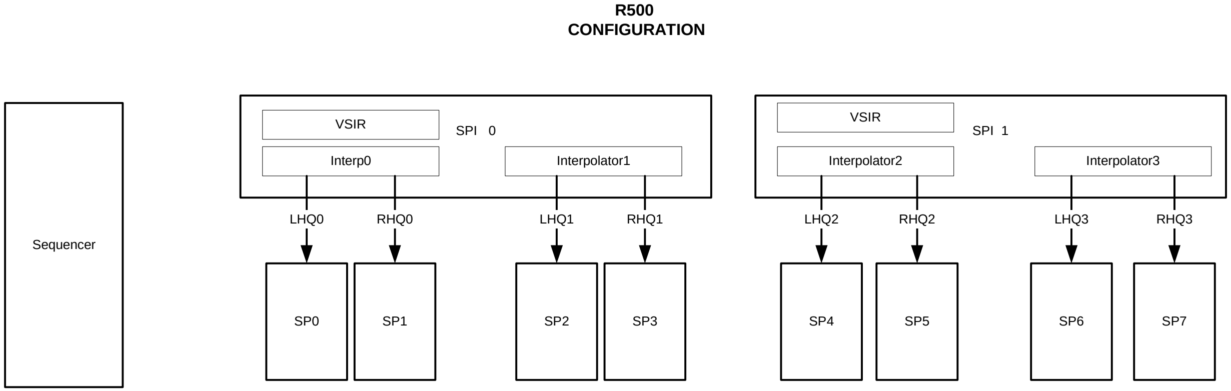


Figure 2: Reservation stations and arbiters

Under this new scheme, the sequencer (SQ) will only use one global state management machine per vector type (pixel, vertex) that we call the reservation station (RS).

1.2 Data Flow graph (SP)



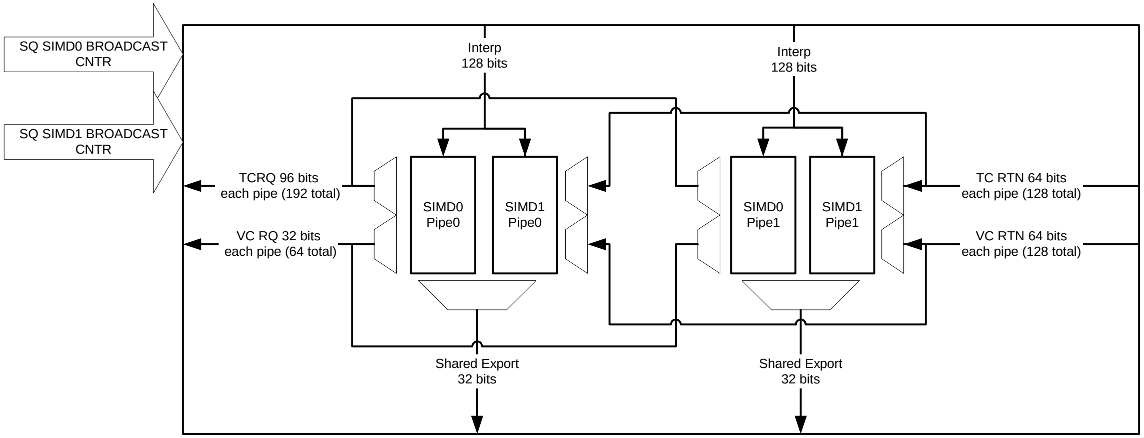


Figure 3: The shader Pipe

1.3 Control Graph

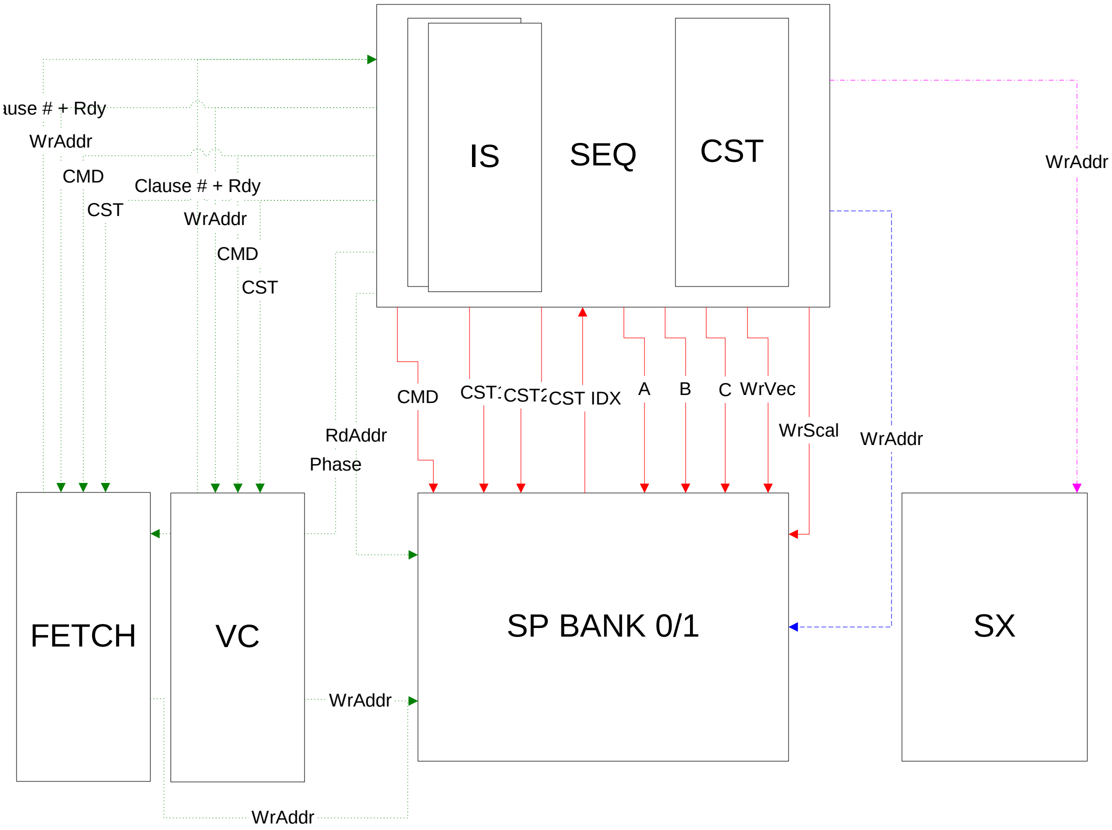


Figure 4: Sequencer Control interfaces

In green is represented the Fetch control interface, in red the ALU control interface, in blue the Interpolated/Vector control interface and in purple is the output file control interface.

2. Interpolated data bus

The interpolators contain an IJ buffer to pack the information as much as possible before writing it to the register file.

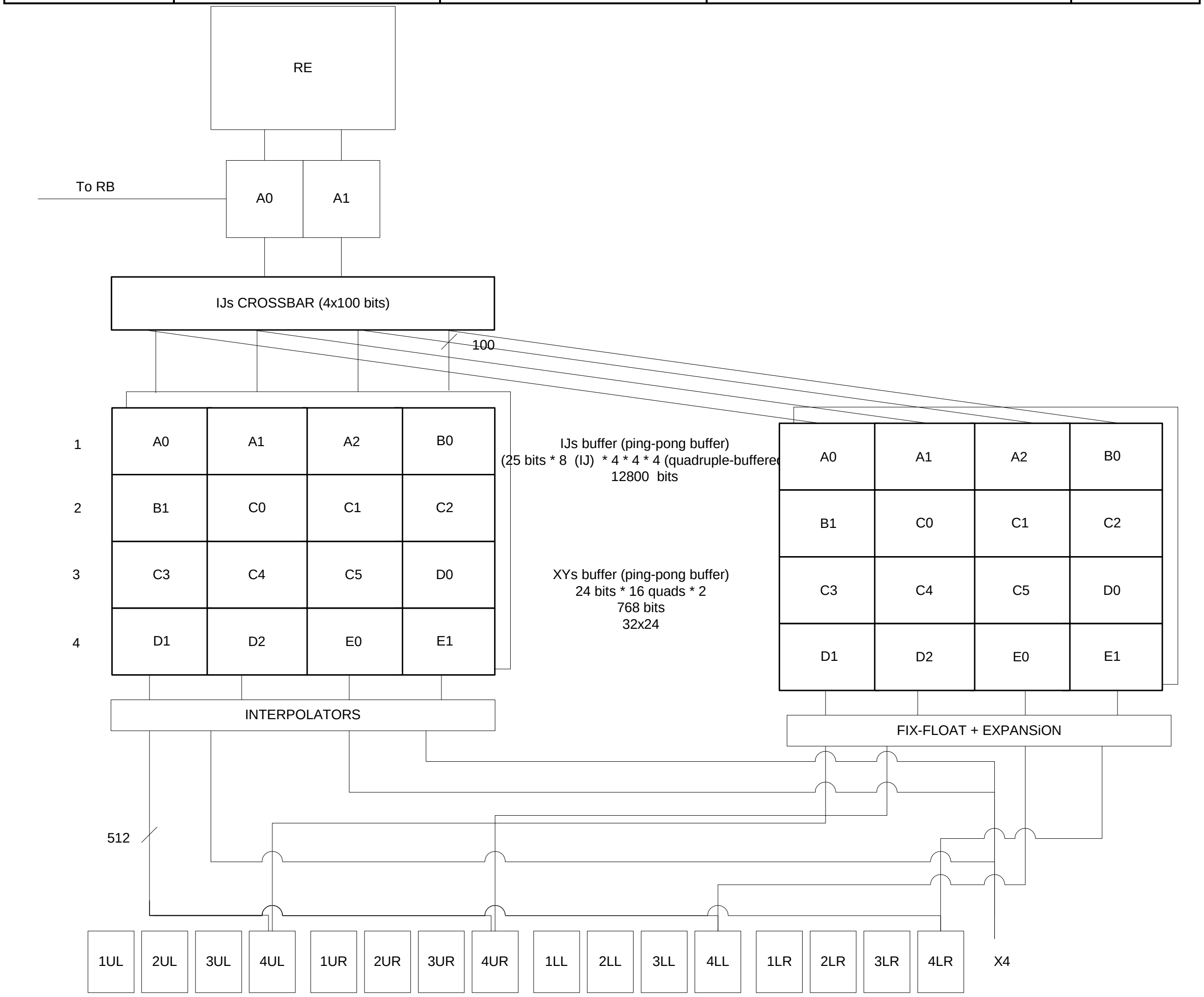
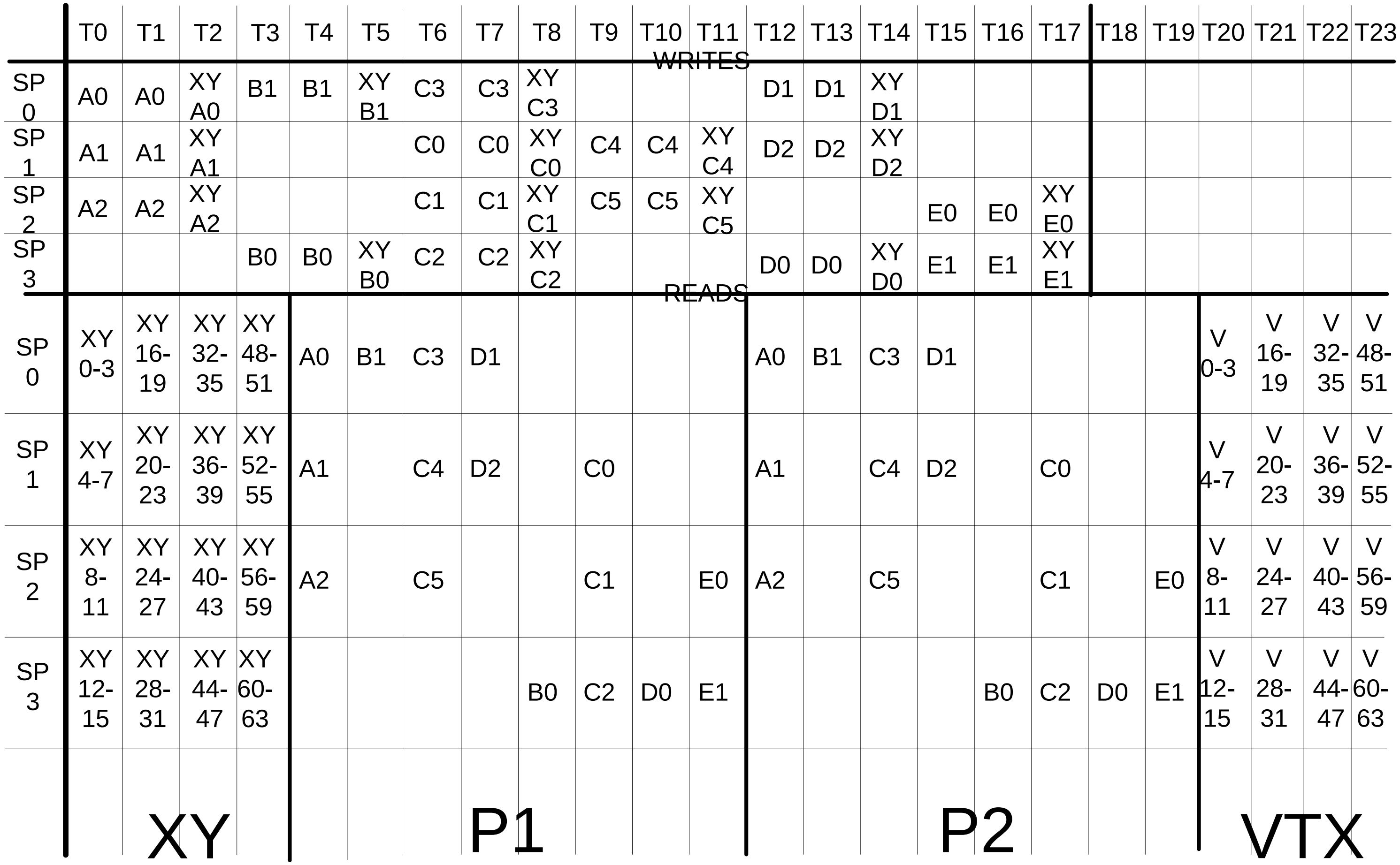


Figure 5: Interpolation buffers



WRITES

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21	T22	T23
SP 0	A0	A0	XY A0	B1	B1	XY B1	C3	C3	XY C3				D1	D1	XY D1									
SP 1	A1	A1	XY A1				C0	C0	XY C0	C4	C4	XY C4	D2	D2	XY D2									
SP 2	A2	A2	XY A2				C1	C1	XY C1	C5	C5	XY C5				E0	E0	XY E0						
SP 3				B0	B0	XY B0	C2	C2	XY C2				D0	D0	XY D0	E1	E1	XY E1						

READS

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21	T22	T23
SP 0	XY 0-3	XY 16-19	XY 32-35	XY 48-51	A0	B1	C3	D1					A0	B1	C3	D1					V 0-3	V 16-19	V 32-35	V 48-51
SP 1	XY 4-7	XY 20-23	XY 36-39	XY 52-55	A1		C4	D2		C0			A1		C4	D2		C0			V 4-7	V 20-23	V 36-39	V 52-55
SP 2	XY 8-11	XY 24-27	XY 40-43	XY 56-59	A2		C5			C1		E0	A2		C5			C1		E0	V 8-11	V 24-27	V 40-43	V 56-59
SP 3	XY 12-15	XY 28-31	XY 44-47	XY 60-63					B0	C2	D0	E1					B0	C2	D0	E1	V 12-15	V 28-31	V 44-47	V 60-63

Figure 6: Interpolation timing diagram

Above is an example of a tile the sequencer might receive from the SC. The write side is how the data get stacked into the XY and IJ buffers, the read side is how the data is passed to the GPRs. The IJ information is packed in the IJ buffer 4 quads at a time or two clocks. The sequencer allows at any given time as many as four quads to interpolate a parameter. They all have to come from the same primitive. Then the sequencer controls the write mask to the GPRs to write the valid data in.

3. Instruction Store

There is going to be two instruction stores for the whole chip. They will each contain 4096 instructions of 96 bits each.

They will be 1 port memories; Ports are allocated in this fashion (but not necessarily in this order):

ALU 0 SIMD0 CF
ALU 0 SIMD0
ALU 1 SIMD0 CF
ALU 1 SIMD0
Fetch CF
Fetch
VC CF
VC

ALU 0 SIMD1 CF
ALU 0 SIMD1
ALU 1 SIMD1 CF
ALU 1 SIMD1
Fetch CF
Fetch
VC CF
VC

Fetch and VC can steal one another's ports with stated resource having priority over its port (this is not really necessary for the R500 but will be for any derivative part because there will only be one instruction store).

Writes are opportunistic.

The instruction store is loaded by the CP thru the register mapped registers.

The VS_BASE and PS_BASE context registers are used to specify for each context where its shader is in the instruction memory.

For the Real time commands the story is quite the same but for some small differences. There are no wrap-around points for real time so the driver must be careful not to overwrite regular shader data. The shared code (shared subroutines) uses the same path as real time.

4. Sequencer Instructions

All control flow instructions instructions are handled by the sequencer only. The ALUs will perform NOPs during this time (MOV PV,PV, PS,PS) if they have nothing else to do.

5. Constant Stores

5.1 Memory organizations

A likely size for the ALU constant store is 1024x128 bits. The read BW from the ALU constant store is 128 bits/clock and the write bandwidth is 32 bits/clock (directed by the CP bus size not by memory ports).

The maximum logical size of the constant store for a given shader is 256 constants. Or 512 for the pixel/vertex shader pair. The size of the re-mapping table is 128 lines (each line addresses 4 constants). The write granularity is 4 constants or 512 bits. It takes 16 clocks to write the four constants. Real time requires 256 lines in the physical memory (this is physically register mapped).

There will be two of those memories and two of each remapping read memories.

The texture state is also kept in a similar memory. The size of this memory is 320x96 bits (128 texture states for regular mode, 32 states for RT). The memory thus holds 128 texture states (192 bits per state). The logical size exposes 32 different states total, which are going to be shared between the pixel and the vertex shader. The size of the re-mapping table to for the texture state memory is 32 lines (each line addresses 1 texture state lines in the real memory). The CP write granularity is 1 texture state lines (or 192 bits). The driver sends 512 bits but the CP ignores the top 320 bits. It thus takes 6 clocks to write the texture state. Real time requires 32 lines in the physical memory (this is physically register mapped).

The control flow constant memory doesn't sit behind a renaming table. It is register mapped and thus the driver must reload its content each time there is a change in the control flow constants. Its size is 320*32 because it must hold 8 copies of the 32 dwords of control flow constants and the loop construct constants must be aligned.

The constant re-mapping tables for texture state and ALU constants are logically register mapped for regular mode and physically register mapped for RT operation.

5.2 Management of the Control Flow Constants

The control flow constants are register mapped, thus the CP writes to the according register to set the constant, the SQ decodes the address and writes to the block pointed by its current base pointer (CF_WR_BASE). On the read side, one level of indirection is used. A register (SQ_CONTEXT_MISC.CF_RD_BASE) keeps the current base pointer to the control flow block. This register is copied whenever there is a state change. Should the CP write to CF after the state change, the base register is updated with the (current pointer number +1)% number of states. This way, if the CP doesn't write to CF the state is going to use the previous CF constants.

5.3 Management of the re-mapping tables

5.3.1 *R400 Constant management*

The sequencer is responsible to manage two re-mapping tables (one for the constant store and one for the texture state). On a state change (by the driver), the sequencer will broadside copy the contents of its re-mapping tables to a new one. We have 8 different re-mapping tables we can use concurrently.

The constant memory update will be incremental, the driver only need to update the constants that actually changed between the two state changes.

For this model to work in its simplest form, the requirement is that the physical memory MUST be at least twice as large as the logical address space + the space allocated for Real Time. In our case, since the logical address space is 512 and the reserved RT space can be up to 256 entries, the memory must be of sizes 1280 and above. Similarly the size of the texture store must be of 32*2+32 = 96 entries and above.

5.3.2 *Dirty bits*

Two sets of dirty bits will be maintained per logical address. The first one will be set to zero on reset and set when the logical address is addressed. The second one will be set to zero whenever a new context is written and set for each address written while in this context. The reset dirty is not set, then writing to that logical address will not require de-allocation of whatever address stored in the renaming table. If it is set and the context dirty is not set, then the physical address store needs to be de-allocated and a new physical address is necessary to store the incoming data. If they are both set, then the data will be written into the physical address held in the renaming for the current logical address. No de-allocation or allocation takes place. This will happen when the driver does a set constant twice to the same logical address between context changes. NOTE: It is important to detect and prevent this, failure to do it will allow multiple writes to allocate all physical memory and thus hang because a context will not fit for rendering to start and thus free up space.

5.3.3 *Free List Block*

A free list block that would consist of a counter (called the IFC or Initial Free Counter) that would reset to zero and incremented every time a chunk of physical memory is used until they have all been used once. This counter would be checked each time a physical block is needed, and if the original ones have not been used up, us a new one, else check the free list for an available physical block address. The count is the physical address for when getting a chunk from the counter.
Storage of a free list big enough to store all physical block addresses.
Maintain three pointers for the free list that are reset to zero. The first one we will call write_ptr. This pointer will identify the next location to write the physical address of a block to be de-allocated. Note: we can never free more

physical memory locations than we have. Once recording address the pointer will be incremented to walk the free list like a ring.
The second pointer will be called stop_ptr. The stop_ptr pointer will be advanced by the number of address chunks de-allocates when a context finishes. The address between the stop_ptr and write_ptr cannot be reused because they are still in use. But as soon as the context using then is dismissed the stop_ptr will be advanced.
The third pointer will be called read_ptr. This pointer will point will point to the next address that can be used for allocation as long as the read_ptr does not equal the stop_ptr and the IFC is at its maximum count.

5.3.4 *De-allocate Block*

This block will maintain a free physical address block count for each context. While in current context, a count shall be maintained specifying how many blocks were written into the free list at the write_ptr pointer. This count will be reset upon reset or when this context is active on the back and different than the previous context. It is actually a count of blocks in the previous context that will no longer be used. This count will be used to advance the write_ptr pointer to make available the set of physical blocks freed when the previous context was done. This allows the discard or de-allocation of any number of blocks in one clock.

5.3.5 *Operation of Incremental model*

The basic operation of the model would start with the write_ptr, stop_ptr, read_ptr pointers in the free list set to zero and the free list counter is set to zero. Also all the dirty bits and the previous context will be initialized to zero. When the first set constants happen, the reset dirty bit will not be set, so we will allocate a physical location from the free list counter because its not at the max value. The data will be written into physical address zero. Both the additional copy of the renaming table and the context zeros of the big renaming table will be updated for the logical address that was written by set start with physical address of 0. This process will be repeated for any logical address that are not dirty until the context changes. If a logical address is hit that has its dirty bits set while in the same context, both dirty bits would be set, so the new data will be over-written to the last physical address assigned for this logical address. When the first draw command of the context is detected, the previous context stored in the additional renaming table will be copied to the larger renaming table in the current (new) context location. Then the set constant logical address with be loaded with a new physical address during the copy and if the reset dirty was set, the physical address it replaced in the renaming table would be entered at the write_ptr pointer location on the free list and the write_ptr will be incremented. The de-allocation counter for the previous context (eight) will be incremented. This as set states come in for this context one of the following will happen:

1.) No dirty bits are set for the logical address being updated. A line will be allocated of the free-list counter or the free list at read_ptr pointer if read_ptr != to stop_ptr .
2.) Reset dirty set and Context dirty not set. A new physical address is allocated, the physical address in the renaming table is put on the free list at write_ptr and it is incremented along with the de-allocate counter for the last context.
3.) Context dirty is set then the data will be written into the physical address specified by the logical address.

This process will continue as long as set states arrive. This block will provide backpressure to the CP whenever he has not free list entries available (counter at max and stop_ptr == read_ptr). The command stream will keep a count of contexts of constants in use and prevent more than max constants contexts from being sent.

Whenever a draw packet arrives, the content of the re-mapping table is written to the correct re-mapping table for the context number. Also if the next context uses less constants than the current one all exceeding lines are moved to the free list to be de-allocated later. This happens in parallel with the writing of the re-mapping table to the correct memory.

Now preferable when the constant context leaves the last ALU clause it will be sent to this block and compared with the previous context that left. (Init to zero) If they differ than the older context will no longer be referenced and thus can be de-allocated in the physical memory. This is accomplished by adding the number of blocks freed this context to the stop_ptr pointer. This will make all the physical addresses used by this context available to the read_ptr allocate pointer for future allocation.

This device allows representation of multiple contexts of constants data with N copies of the logical address space. It also allows the second context to be represented as the first set plus some new additional data by just storing the delta's. It allows memory to be efficiently used and when the constants updates are small it can store multiple context. However, if the updates are large, less contexts will be stored and potentially performance will be degraded. Although it will still perform as well as a ring could in this case.

5.4 Constant Store Indexing

In order to do constant store indexing, the sequencer must be loaded first with the indexes (that come from the GPRs). There are 144 wires from the exit of the SP to the sequencer (9 bits pointers x 16 vertexes/clock).

```
MOVA  R1.X,R2.X        // Loads the sequencer with the content of R2.X, also copies the content of R2.X into R1.X
ADD   R3,R4,C0[R2.X]// Uses the state from the sequencer to add R4 to C0[R2.X] into R3
```

Note that we don't really care about what is in the brackets because we use the state from the MOVA instruction. R2.X is just written again for the sake of simplicity and coherency.

The storage needed in the sequencer in order to support this feature is 2*64*9 bits = 1152 bits.

The address register is a signed integer, which ranges from –256 to 255.

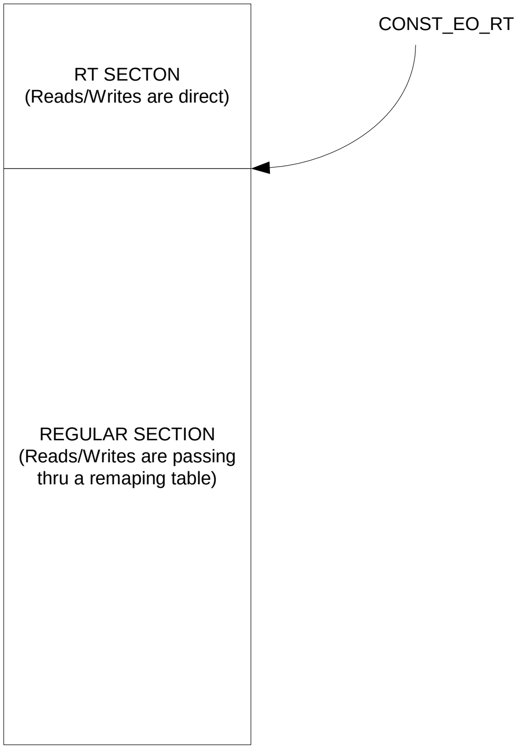
The address register is not kept across clause boundaries. As such, it must be refreshed after any Serialize (or yield), allocate instruction or resource change. Failure to refresh the address register will result in unpredictable behavior.

5.5 Real Time Commands

The real time commands constants are written by the CP using the register mapped registers allocated for RT. It works is the same way than when dealing with regular constant loads BUT in this case the CP is not sending a logical address but rather a physical address and the reads are not passing thru the re-mapping table but are directly read from the memory. The boundary between the two zones is defined by the CONST_EO_RT control register. Similarly, for the fetch state, the boundary between the two zones is defined by the TSTATE_EO_RT control register.

5.6 Constant Waterfalling

In order to have a reasonable performance in the case of constant store indexing using the address register, we are going to have the possibility of using the physical memory port for read only. This way we can read 1 constant per clock and thus have a worst-case waterfall mode of 1 vertex per clock. There is a small synchronization issue related with this as we need for the SQ to make sure that the constants where actually written to memory (not only sent to the sequencer) before it can allow the first vector of pixels or vertices of the state to go thru the ALUs. To do so, the sequencer keeps 8 bits (one per render state) and sets the bits whenever the last render state is written to memory and clears the bit whenever a state is freed.



CONST_EO_RT
RT SECTON
(Reads/Writes are direct)
REGULAR SECTION
(Reads/Writes are passing thru a remaping table)

Figure 7: The Constant store

6. Looping and Branches

Loops and branches are planned to be supported and will have to be dealt with at the sequencer level. We plan on supporting constant loops and branches using a control program.

6.1 The controlling state.

The R400 controling state consists of:

Boolean[255:0]
Loop_count[7:0][31:0]
Loop_Start[7:0][31:0]
Loop_Step[7:0][31:0]

That is 256 Booleans and 32 loops.

We have a stack of 4 elements for nested calls of subroutines and 4 loop counters to allow for nested loops.

This state is available on a per shader program basis.

6.2 The Control Flow Program

We'd like to be able to code up a program of the form:

```
1:      Loop
2:      Exec    TexFetch
```

3: TexFetch
4: ALU
5: ALU
6: TexFetch
7: End Loop
8: ALU Export

But realize that 3: may be dependent on 2: and 4: is almost certainly dependent on 2: and 3:. Without clausing, these dependencies need to be expressed in the Control Flow instructions. Additionally, without separate 'texture clauses' and 'ALU clauses' we need to know which instructions to dispatch to the Texture Unit and which to the ALU unit. This information will be encapsulated in the flow control instructions.

Each control flow instruction will contain 2 bits of information for each (non-control flow) instruction:
a) ALU or Texture
b) Serialize Execution

(b) would force the thread to stop execution at this point (before the instruction is executed) and wait until all textures have been fetched. Given the allocation of reserved bits, this would mean that the count of an 'Exec' instruction would be limited to about 8 (non-control-flow) instructions. If more than this were needed, a second Exec (with the same conditions) would be issued.

Another function that relies upon 'clauses' is allocation and order of execution. We need to assure that pixels and vertices are exported in the correct order (even if not all execution is ordered) and that space in the output buffers are allocated in order. Additionally data can't be exported until space is allocated. A new control flow instruction:

Alloc <buffer select -- position,parameter, pixel or vertex memory. And the size required>.

would be created to mark where such allocation needs to be done. To assure allocation is done in order, the actual allocation for a given thread can not be performed unless the equivalent allocation for all previous threads is already completed. The implementation would also assure that execution of instruction(s) following the serialization due to the Alloc will occur in order -- at least until the next serialization or change from ALU to Texture. In most cases this will allow the exports to occur without any further synchronization. Only 'final' allocations or position allocations are guaranteed to be ordered. Because strict ordering is required for pixels, parameters and positions, this implies only a single alloc for these structures. Vertex exports to memory do not require ordering during allocation and so multiple 'allocs' may be done.

6.2.1 *Control flow instructions table*

Here is the revised control flow instruction set.

Note that whenever a field is marked as RESERVED, it is assumed that all the bits of the field are cleared (0).

NOP		
47 … 44	43	42 … 0
0000	Addressing	RESERVED

This is a regular NOP.

Execute						
47 … 44	43	40 … 34	33 … 28	27 …16	15…12	11 … 0
0001	Addressing	RESERVED	Vertex Cache	Instructions type + serialize (6 instructions)	Count	Exec Address

Execute_End						
47 … 44	43	40 … 34	33 … 28	27 …16	15…12	11 … 0
0010	Addressing	RESERVED	Vertex Cache	Instructions type + serialize (6 instructions)	Count	Exec Address

Execute up to 6 instructions at the specified address in the instruction memory. The Instruction type field tells the sequencer the type of the instruction (LSB) (1 = Texture, 0 = ALU and whether to serialize or not the execution (MSB) (1 = Serialize, 0 = Non-Serialized). If the corresponding VC bit is set then VC is used instead of TP/ALU. If Execute_End this is the last execution block of the shader program.

Vertex Cache	Serialize	Instruction Type (Resource)	
0	0	0	: ALU instruction, not yielding
0	0	1	: Texture instruction, not yielding
0	1	0	: ALU instruction, yielding
0	1	1	: Texture instruction, yielding
1	0	0	: Vertex cache instruction, not yielding
1	0	1	: Vertex cache instruction, not yielding
1	1	0	: Vertex cache instruction, yielding
1	1	1	: Vertex cache instruction, yielding

Conditional_Execute							
47 … 44	43	42	41 … 34	33…28	27…16	15 …12	11 … 0
0011	Addressing	Condition	Boolean address	Vertex Cache	Instructions type + serialize (6 instructions)	Count	Exec Address

Conditional_Execute_End							
47 … 44	43	42	41 … 34	33…28	27…16	15 …12	11 … 0
0100	Addressing	Condition	Boolean address	Vertex Cache	Instructions type + serialize (6 instructions)	Count	Exec Address

If the specified Boolean (8 bits can address 256 Booleans) meets the specified condition then execute the specified instructions (up to 9 instructions). If the condition is not met, we go on to the next control flow instruction. If Conditional_Execute_End and the condition is met, this is the last execution block of the shader program.

Conditional_Execute_Predicates								
47 … 44	43	42	41 … 36	35 … 34	33…28	27…16	15…12	11 … 0
0101	Addressing	Condition	RESERVED	Predicate vector	Vertex Cache	Instructions type + serialize (6 instructions)	Count	Exec Address

Conditional_Execute_Predicates_End								
47 … 44	43	42	41 … 36	35 … 34	33…28	27…16	15…12	11 … 0
0110	Addressing	Condition	RESERVED	Predicate vector	Vertex Cache	Instructions type + serialize (6 instructions)	Count	Exec Address

Check the AND/OR of all current predicate bits. If AND/OR matches the condition execute the specified number of instructions. We need to AND/OR this with the kill mask in order not to consider the pixels that aren't valid. If the condition is not met, we go on to the next control flow instruction. If Conditional_Execute_Predicates_End and the condition is met, this is the last execution block of the shader program.

Conditional_Execute_Predicates_No_Stall								
47 … 44	43	42	41 … 36	35 … 34	33…28	27…16	15…12	11 … 0
1101	Addressing	Condition	RESERVED	Predicate vector	Vertex Cache	Instructions type + serialize (6 instructions)	Count	Exec Address

Conditional_Execute_Predicates_No_Stall_End								
47 … 44	43	42	41 … 36	35 … 34	33…28	27…16	15…12	11 … 0
1110	Addressing	Condition	RESERVED	Predicate vector	Vertex Cache	Instructions type + serialize (6 instructions)	Count	Exec Address

Same as Conditionnal_Execute_Predicates but the SQ is not going to wait for the predicate vector to be updated. You can only set this in the compiler if you know that the predicate set is only a refinement of the current one (like a nested if) because the optimization would still work.

Loop_Start					
47 … 44	43	42 … 21	20 … 16	15…13	12 … 0
0111	Addressing	RESERVED	loop ID	RESERVED	Jump address

Loop Start. Compares the loop iterator with the end value. If loop condition not met jump to the address. Forward jump only. Also computes the index value. The loop id must match between the start to end, and also indicates which control flow constants should be used with the loop.

Loop_End									
47 …44	43	42	41… 36	35…34	33… 22	21	20 … 16	15…13	12 … 0
1000	Addressing	Cond	RESERVED	Predicate Vector	RESERVED	Pred break	loop ID	RESERVED	start address

Loop end. Increments the counter by one, compares the loop count with the end value. If loop condition met, continue, else, jump BACK to the start of the loop. If predicate break != 0, then compares predicate vector n (specified by predicate Vector) to condition. If all bits meet condition then break the loop.

The way this is described does not prevent nested loops, and the inclusion of the loop id make this easy to do.

Conditionnal_Call						
47 … 44	43	42	41 … 34	33 … 14	13	12 … 0
1001	Addressing	Condition	Boolean address	RESERVED	Force Call	Jump address

If the condition is met, jumps to the specified address and pushes the control flow program counter on the stack. If force call is set the condition is ignored and the call is made always.

Return		
47 … 44	43	42 … 0
1010	Addressing	RESERVED

Pops the topmost address from the stack and jumps to that address. If nothing is on the stack, the program will just continue to the next instruction.

Conditionnal_Jump							
47 … 44	43	42	41… 34	33	32 … 14	13	12 … 0
1011	Addressing	Condition	Boolean address	FW only	RESERVED	Force Jump	Jump address

If force jump is set the condition is ignored and the jump is made always. If FW only is set then only forward jumps are allowed.

Allocate					
47 … 44	43	42…41	40	39 … 3	2…0
1100	Debug	Buffer Select	No Serial	RESERVED	Size

Buffer Select takes a value of the following:
01 – position export (ordered export)
10 – parameter cache or pixel export (ordered export)
11 – pass thru (out of order exports).

Size field is only used to reserve space in the export buffer for pass thru exports. Valid values are 1 (1 line) thru 5 (5 lines). It should be determined by the compiler/assembler by taking max index used +1.

If debug is set this is a debug alloc (ignore if debug DB_ON register is set to off).

By default the serial bit is set on an alloc. If the No Serial bit is asserted then the serial bit won't be set in the SQ.

6.2.2 *Alloc Statements*

Alloc statements are control flow instructions that allocate resources that are required for executable export instructions. An alloc statement can be either a normal yield point, or a partial yield. At a partial yield - hardware releases the gpu so all state (mova, grad etc) is lost but the thread can resume before all pending fetches have completed.

There are three types of allocs:

alloc-position - proceeds the export of position from a vertex shader. A vertex shader must include one alloc of position. A position alloc cannot be used in a pixel shader. all exports for position must execute between the alloc-position and the next yield point or resource change. There is a small performance advantage to placing the alloc-position near the top of the vertex shader. However we don't think this is worth adding an
extra instruction or register to the shader.

alloc-interp/color - proceeds interpolator exports in a vertex shader or the color exports in a pixel shader. There can be only one alloc interp/color per shader. The color alloc in a pixel shader must be after any alloc-mem-exports. The actual exports can occur anywhere between the alloc-interp/color and the end of the program. There is a small performance advantage to placing the alloc-interp/color near the bottom of the shader. However we don't think this is worth adding an extra instruction or register to the shader. There is a big performance advantage of having no fetches of any kind after the alloc-interp/color.

alloc-mem-export - proceeds any memory-address, memory-data exports. There can be multiple alloc-mem-export statements in either kind of shader. All exports for mem-exports must execute between the corresponding alloc-mem-export and the next yield point or resource change.

6.3 Implementation

The envisioned implementation has a buffer that maintains the state of each thread. A thread lives in a given location in the buffer during its entire life, but the buffer has FIFO qualities in that threads leave in the order that they enter. Actually two buffers are maintained -- one for Vertices and one for Pixels. The intended implementation would allow for:

16 entries for vertices
48 entries for pixels.

From each buffer, arbitration logic attempts to select 1 thread for the texture unit and 2 (interleaved) thread for the ALU unit. Once a thread is selected it is read out of the buffer, marked as invalid, and submitted to appropriate execution unit. It is returned to the buffer (at the same place) with its status updated once all possible sequential instructions have been executed. A switch from ALU to TEX or visa-versa or a Serialize_Execution modifier forces the thread to be returned to the buffer.

Each entry in the buffer will be stored across two physical pieces of memory - most bits will be stored in a 1 read port device. Only bits needed for thread arbitration will be stored in a highly multi-ported structure. The bits kept in the 1 read port device will be termed 'state'. The bits kept in the multi-read ported device will be termed 'status'.

'State Bits' needed include:

1. Control Flow Instruction Pointer (13 bits),
2. Execution Count Marker 4 bits),
3. Loop Iterators (4x9 bits),
4. Loop Counters (4x9 bits),
5. Call return pointers (4x13 bits),
6. Predicate Bits (64 bits),
7. Export ID (4 bits),
8. Parameter Cache base Ptr (7 bits),
9. GPR Base Ptr (8 bits),
10. Context Ptr (3 bits).
11. LOD corrections (6x16 bits)
12. Valid bits (64 bits)
13. RT (1 bit) Signifies that this thread is a Real Time thread. This bit must be sent to the Constant store state machine when reading it.

Absent from this list are 'Index' pointers. These are costly enough that I'm presuming that they are instead stored in the GPRs. The first seven fields above (Control Flow Ptr, Execution Count, Loop Counts, call return ptrs, Predicate bits, PC base ptr and export ID) are updated every time the thread is returned to the buffer based on how much progress has been mode on thread execution. GPR Base Ptr, Context Ptr and LOD corrections are unchanged throughout execution of the thread.

'Status Bits' needed include:

- Valid Thread
- ALU engine needed
- Texture engine needed
- VC engine needed
- Texture Reads are outstanding
- VC Reads are outstanding
- Alu bank (0/1)
- Waiting on Texture Read to Complete
- Allocation Wait (2 bits)
- 00 – No allocation needed
- 01 – Position export allocation needed (ordered export)
- 10 – Parameter or pixel export needed (ordered export)
- 11 – pass thru (out of order export)
- Allocation Size (4 bits)
- Position Allocated
- Mem/Color Allocated
- First thread of a new context
- Event thread (NULL thread that needs to trickle down the pipe)
- Last (1 bit)
- Pulse SX (1 bit)

All of the above fields from all of the entries go into the arbitration circuitry. The arbitration circuitry will select a winner for both the Texture Engine and for the ALU engine. There are actually two sets of arbitration -- one for pixels and one for vertices. A final selection is then done between the two. But the rest of this implementation summary only considers the 'first' level selection which is similar for both pixels and vertices.

Texture arbitration requires no allocation or ordering so it is purely based on selecting the 'oldest' thread that requires the Texture Engine.

ALU arbitration is a little more complicated. First, only threads where either of Texture_Reads_outstanding or Waiting_on_Texture_Read_to_Complete are '0' are considered. Then if Allocation_Wait is active, these threads are further filtered based on whether space is available. If the allocation is position allocation, then the thread is only considered if all 'older' threads have already done their position allocation (position allocated bits set). If the allocation is parameter or pixel allocation, then the thread is only considered if it is the oldest thread. Also a thread is not considered if it is a parameter or pixel or position allocation, has its First_thread_of_a_new_context bit set and would cause ALU interleaving with another thread performing the same parameter or pixel or position allocation. Finally the 'oldest' of the threads that pass through the above filters is selected. If the thread needed to allocate, then at this time the allocation is done, based on Allocation_Size. If a thread has its “last” bit set, then it is also removed from the buffer, never to return.

If I now redefine 'clauses' to mean 'how many times the thread is removed from the thread buffer for the purpose of exection by either the ALU or Texture engine', then the minimum number of clauses needed is 2 -- one to perform the allocation for exports (execution automatically halts after an 'Alloc' instruction) (but doesn't performs the actual allocation) and one for the actual ALU/export instructions. As the 'Alloc' instruction could be part of a texture clause (presumably the final instruction in such a clause), a thread could still execute in this minimal number of 2 clauses, even if it involved texture fetching.

The Texture_Reads_Outstanding and VC_reads_Outstanding bits tell the SQ that a texture or VC read is outstanding. In this case, if we encounter a serial bit we need to wait until both resources are free (pending = 0) in order to proceed.

6.4 Data dependant predicate instructions

Data dependant conditionals will be supported in the R400. The only way we plan to support those is by supporting three vector/scalar predicate operations of the form:

PRED_SETE_PUSH - similar to SETE except that the result is 'exported' to the sequencer.
PRED_SETNE_PUSH - similar to SETNE except that the result is 'exported' to the sequencer.
PRED_SETGT_PUSH - similar to SETGT except that the result is 'exported' to the sequencer
PRED_SETGTE_PUSH - similar to SETGTE except that the result is 'exported' to the sequencer

For the scalar operations only we will also support the two following instructions:
PRED_SETE
PRED_SETNE
PRED_SETGT
PRED_SET_INV
PRED_SET_POP
PRED_SET_CLR
PRED_SET_RESTORE

Details about actual implementation of these opcodes are in the shader pipe architectural spec.

The export is a single bit - 1 or 0 that is sent using the same data path as the MOVA instruction. The sequencer will maintain 1 set of 64 bits predicate vectors (in fact 2 sets because we interleave two programs but only 1 will be exposed) and use it to control the write masking. This predicate is maintained across clause boundaries.

Then we have two conditional execute bits. The first bit is a conditional execute “on” bit and the second bit tells us if we execute on 1 or 0. For example, the instruction:

P0_ ADD_# R0,R1,R2

Is only going to write the result of the ADD into those GPRs whose predicate bit is 0. Alternatively, P1_ADD_# would only write the results to the GPRs whose predicate bit is set. The use of the P0 or P1 without precharging the sequencer with a PRED instruction is undefined.

6.5 HW Detection of PV,PS

Because of the control program, the compiler cannot detect statically dependant instructions. In the case of non-masked writes and subsequent reads the sequencer will insert uses of PV,PS as needed. This will be done by comparing the read address and the write address of consecutive instructions. For masked writes, the sequencer will insert detect wich channels to read from the GPRs and which ones to read from the PV/PS.

6.6 Register file indexing

Because we can have loops in fetch clause, we need to be able to index into the register file in order to retrieve the data created in a fetch clause loop and use it into an ALU clause. The instruction will include the base address for register indexing and the instruction will contain these controls:

Bit7	Bit 6	
0	0	'absolute register'
0	1	'relative register'
1	0	'previous vector'
1	1	'previous scalar'

In the case of an absolute register we just take the address as is. In the case of a relative register read we take the base address and we add to it the loop_index and this becomes our new address that we give to the shader pipe.

The sequencer is going to keep a loop index computed as such:

Index = Loop_iterator*Loop_step + Loop_start.

We loop until loop_iterator = loop_count. Loop_step is a signed value [-128…127]. The computed index value is a 10 bit counter that is also signed. Its real range is [-256,256]. The tenth bit is only there so that we can provide an out of range value to the “indexing logic” so that it knows when the provided index is out of range and thus can make the necessary arrangements.

6.7 Debugging the Shaders

In order to be able to debug the pixel/vertex shaders efficiently, we provide 2 methods.

6.7.1 *Method 1: Debugging registers*

Current plans are to expose 2 debugging, or error notification, registers:
1. address register where the first error occurred
2. count of the number of errors

The sequencer will detect the following groups of errors:
- count overflow
- constant indexing overflow
- register indexing overflow

Compiler recognizable errors:
- jump errors
    relative jump address > size of the control flow program
- call stack
    call with stack full
    return with stack empty

With all the other errors, program can continue to run, potentially to worst-case limits.

If indexing outside of the constant or the register range, causing an overflow error, the hardware is specified to return the value with an index of 0. This could be exploited to generate error tokens, by reserving and initializing the 0th register (or constant) for errors.

{ISSUE : Interrupt to the driver or not?}

6.7.2 *Method 2: Exporting the values in the GPRs*

1) The sequencer will have a debug active, count register and an address register for this mode.

Under the normal mode execution follows the normal course.

Under the debug mode it is assumed that the program is always exporting n debug vectors and that all other exports to the SX block (but for position) will be turned off (changed into NOPs) by the sequencer (even if they occur before the address stated by the ADDR debug register).

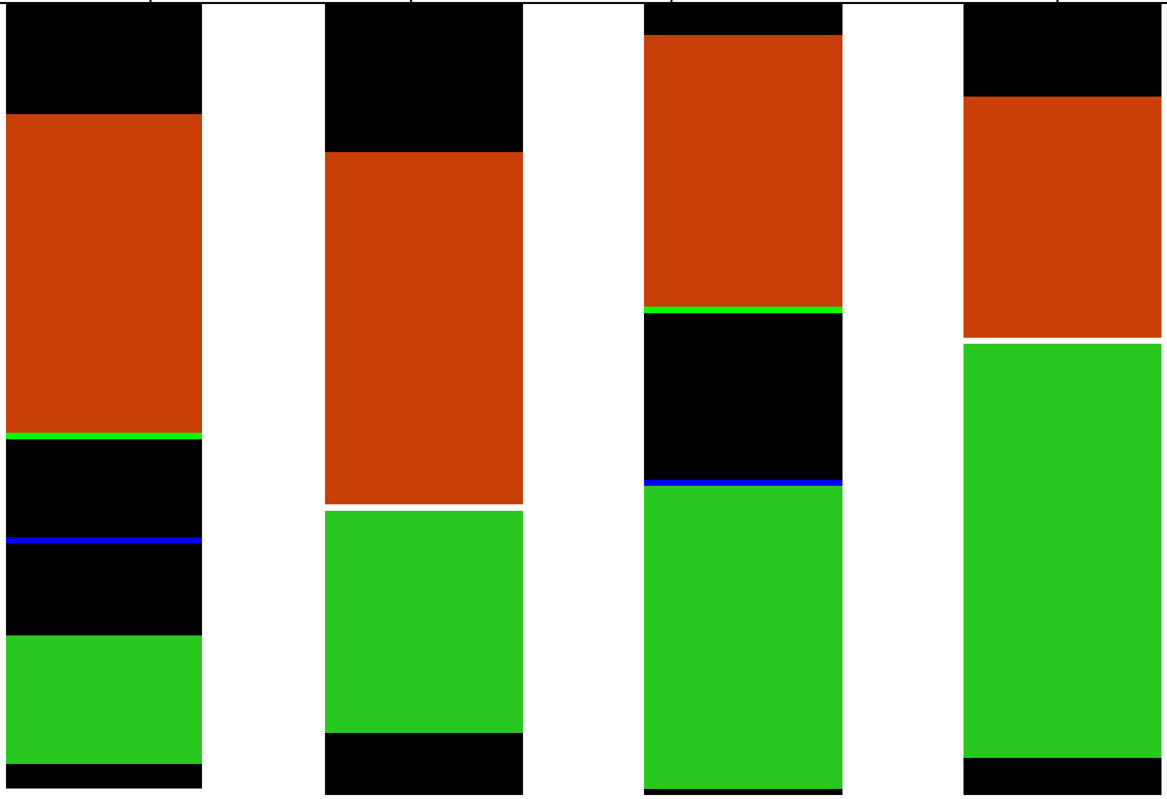
7. Pixel Kill Mask

A vector of 64 bits is kept by the sequencer per group of pixels/vertices. Its purpose is to optimize the texture fetch requests and allow the shader pipe to kill pixels using the following instructions:

MASK_SETE
MASK_SETNE
MASK_SETGT
MASK_SETGTE

8. Register file allocation

The register file allocation for vertices and pixels can either be static or dynamic. In both cases, the register file in managed using two round robins (one for pixels and one for vertices). In the dynamic case the boundary between pixels and vertices is allowed to move, in the static case it is fixed to 128-VERTEX_REG_SIZE for vertices and PIXEL_REG_SIZE for pixels.



Above is an example of how the algorithm works. Vertices come in from top to bottom; pixels come in from bottom to top. Vertices are in orange and pixels in green. The blue line is the tail of the vertices and the green line is the tail of the pixels. Thus anything between the two lines is shared. When pixels meets vertices the line turns white and the boundary is static until both vertices and pixels share the same “unallocated bubble”. Then the boundary is allowed to move again. The numbering of the GPRs starts from the bottom of the picture at index 0 and goes up to the top at index 127.

9. Fetch Arbitration

The fetch arbitration logic chooses one of the n potentially pending fetch clauses to be executed. The choice is made by looking at the Vs and Ps reservation stations and picking the first one ready to execute. Once chosen, the clause state machine will send one 2x2 fetch per clock (or 4 fetches in one clock every 4 clocks) until all the fetch instructions of the clause are sent. This means that there cannot be any dependencies between two fetches of the same clause.

The arbitrator will not wait for the fetches to return prior to selecting another clause for execution. The fetch pipe will be able to handle up to X(?) in flight fetches and thus there can be a fair number of active clauses waiting for their fetch return data.

10. VC Arbitration

The VC arbitration logic chooses one of the n potentially pending VC clauses to be executed. The choice is made by looking at the Vs and Ps reservation stations and picking the first one ready to execute. Once chosen, the clause state machine will send one 2x2 fetch per clock (or 4 fetches in one clock every 4 clocks) until all the fetch instructions of the clause are sent. This means that there cannot be any dependencies between two fetches of the same clause.

The arbitrator will not wait for the fetches to return prior to selecting another clause for execution. The VC pipe will be able to handle up to X(?) in flight VC fetches and thus there can be a fair number of active clauses waiting for their fetch return data.

11. ALU Arbitration

ALU arbitration proceeds in almost the same way than fetch arbitration. The ALU arbitration logic chooses one of the n potentially pending ALU clauses to be executed. The choice is made by looking at the Vs and Ps reservation stations and picking the first one ready to execute. There are two ALU arbiters, one for the even clocks and one for the odd clocks. For example, here is the sequencing of two interleaved ALU clauses (E and O stands for Even and Odd sets of 4 clocks):

Einst0 Oinst0 Einst1 Oinst1 Einst2 Oinst2 Einst0 Oinst3 Einst1 Oinst4 Einst2 Oinst0…
Proceeding this way hides the latency of 8 clocks of the ALUs. Also note that the interleaving also occurs across clause boundaries.

12. Handling Stalls

When the output file is full, the sequencer prevents the ALU arbitration logic from selecting the last clause (this way nothing can exit the shader pipe until there is place in the output file. If the packet is a vertex packet and the position buffer is full (POS_FULL) then the sequencer also prevents a thread from entering an exporting clause. The sequencer will set the OUT_FILE_FULL signal n clocks before the output file is actually full and thus the ALU arbiter will be able read this signal and act accordingly by not preventing exporting clauses to proceed.

12.1 SP stall conditions

12.1.1 *PS Stalls*

None.

12.1.2 *PV Stalls*

None.

13. Content of the reservation station FIFOs

The reservation FIFOs contain the state of the vector of pixels and vertices. We have two sets of those: one for pixels, and one for vertices. They contain 3 bits of Render State 7 bits for the base address of the GPRs, some bits for LOD correction and coverage mask information in order to fetch fetch for only valid pixels, the quad address.

14. The Output File

The output file is where pixels are put before they go to the RBs. The write BW to this store is 256 bits/clock. Just before this output file are staging registers with write BW 512 bits/clock and read BW 256 bits/clock. The staging registers are 4x128 (and there are 16 of those on the whole chip).

15. IJ Format

The IJ information sent by the PA is of this format on a per quad basis:

We have a vector of IJ’s (one IJ per pixel at the centroid of the fragment or at the center of the pixel depending on the mode bit). All pixel’s parameters are always interpolated at full 20x24 mantissa precision.

$$P0 = A + I(0) * (B - A) + J(0) * (C - A)$$
$$P1 = A + I(1) * (B - A) + J(1) * (C - A)$$
$$P2 = A + I(2) * (B - A) + J(2) * (C - A)$$
$$P3 = A + I(3) * (B - A) + J(3) * (C - A)$$

P0	P1
P2	P3

Multiplies (Full Precision): 8
Subtracts 19x24 (Parameters): 2
Adds: 8

FORMAT OF P's IJ : Mantissa 20 Exp 4 for I + Sign
Mantissa 20 Exp 4 for J + Sign

Total number of bits : 20*8 + 4*8 + 4*2 = 200.

All numbers are kept using the un-normalized floating point convention: if exponent is different than 0 the number is normalized if not, then the number is un-normalized. The maximum range for the IJs (Full precision) is +/- 1024.

15.1 Interpolation of constant attributes

Because of the floating point imprecision, we need to take special provisions if all the interpolated terms are the same or if two of the terms are the same.

16. Staging Registers

In order for the reuse of the vertices to be 14, the sequencer will have to re-order the data sent IN ORDER by the VGT for it to be aligned with the parameter cache memory arrangement. Given the following group of vertices sent by the VGT:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 || 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 || 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 || 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63

The sequencer will re-arrange them in this fashion:

0 1 2 3 16 17 18 19 32 33 34 35 48 49 50 51 || 4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55 || 8 9 10 11 24 25 26 27 40 41 42 43 56 57 58 59 || 12 13 14 15 28 29 30 31 44 45 46 47 60 61 62 63

The || markers show the SP divisions. In the event a shader pipe is broken, the SQ is responsible to insert padding to account for the missing pipe. For example, if SP1 is broken, vertices 4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55 will not be sent by the VGT to the SQ **AND** the SQ is responsible to "jump" over these vertices in order for no valid vertices to be sent to an invalid SP.

The most straightforward, *non-compressed* interface method would be to convert, in the VGT, the data to 32-bit floating point prior to transmission to the VSISRs. In this scenario, the data would be transmitted to (and stored in) the VSISRs in full 32-bit floating point. This method requires three 24-bit fixed-to-float converters in the VGT. Unfortunately, it also requires and additional 3,072 bits of storage across the VSISRs. This interface is illustrated in Figure 9. The area of the fixed-to-float converters and the VSISRs for this method is roughly estimated as 0.759sqmm using the R300 process. The gate count estimate is shown in Figure 8.

Basis for 8-deep Latch Memory (from R300)

8x24-bit 11631 μ^2 60.57813 μ^2 per bit

Area of 96x8-deep Latch Memory 46524 μ^2

Area of 24-bit Fix-to-float Converter 4712 μ^2 per converter

Method 1

Block	Quantity	Area
F2F	3	14136
8x96 Latch	16	744384
		758520 μ^2

Figure 8:Area Estimate for VGT to Shader Interface

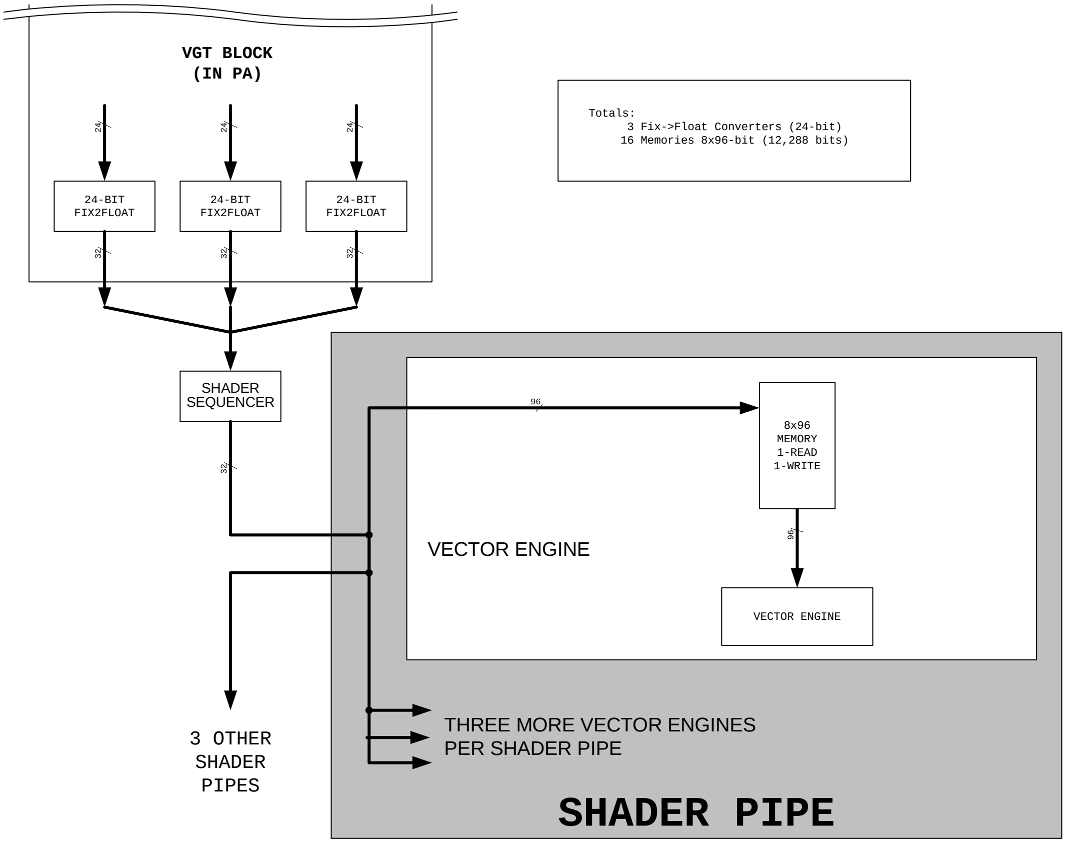


Figure 9:VGT to Shader Interface

17. The parameter cache

The parameter cache is where the vertex shaders export their data. It consists of 16 128x128 memories (1R/1W). The reuse engine will make it so that all vertexes of a given primitive will hit different memories. The allocation method for these memories is a simple round robin. The parameter cache pointers are mapped in the following way: 4MSBs are the memory number and the 7 LSBs are the address within this memory.

MEMORY NUMBER 4 bits	ADDRESS 7 bits

The PA generates the parameter cache addresses as the positions come from the SQ. All it needs to do is keep a Current_Location pointer (7 bits only) and as the positions comes increment the memory number. When the memory number field wraps around, the PA increments the Current_Location by VS_EXPORT_COUNT (a snooped register from the SQ). As an example, say the memories are all empty to begin with and the vertex shader is exporting 8 parameters per vertex (VS_EXPORT_COUNT = 8). The first position received is going to have the PC address 00000000000 the second one 00010000000, third one 00100000000 and so on up to 11110000000. Then the next position received (the 17th) is going to have the address 00000001000, the 18th 00010001000, the 19th 00100001000 and so on. The Current_location is NEVER reset BUT on chip resets. The only thing to be careful about is that if the SX doesn't send you a full group of positions (<64) then you need to fill the address space so that the next group starts correctly aligned (for example if you receive only 33 positions then you need to add 2*VS_EXPORT_COUNT to Current_Location and reset the memory count to 0 before the next vector begins).

17.1 Export restrictions

17.1.1 *Pixel exports:*

Pixels can export 1,2,3 or 4 color buffers to the SX(+z). The exports will be done in order. The exports will always be ordered to the SX.

17.1.2 *Vertex exports:*

Position or parameter caches can be exported in any order in the shader program. It is always better to export posistion as soon as possible. Position has to be exported in a single export block (no texture instructions can be placed between the exports). Parameter cache exports can be done in any order with texture instructions interleaved. The exports will always be allocated in order to the SX.

17.1.3 *Pass thru exports:*

Pass thru exports have to be done in groups of the form:

```
Alloc 1 thru 5 (max export offset + 1, for example if using EM4 alloc size 5)
Execute ALU(ADDR) ALU(DATA) ALU(DATA) ALU(DATA)…
```

When exporting to more than EM0, one MUST write to EM4 also (the write may be predicated if you don't need the export). This is used to initialize the buffers in the SX.

There cannot be any serialize bits set OR texture Reads between the EA and the last EM.

Memory exports will be surfaced using a macro extension; here is what needs to happen inside the macro:

The macro needs to create a special constant of the form:

Stream ID constant:
.x = Integer that holds BaseAddressInBytes/4 in bits (29:0). Bits 31:30 should be 0b01.
.y = 2**23
.z = Integer that holds register field data. Note that this data must be organized so that it always represents a 'valid' floating point number, with the relevant bits in (23 - 0); One way of doing this would be to take the 23 bits and add 2**23.
.w = max index value + 2**23

Output to EXaddress:

.x = Base of array (in low 30 bits)/4

.y	= Index value (in low 23 bits)
.z	= Register Field data (in low 23 bits)
.w	= Max Index value (in low 23 bits)

Also Assume that C0:

.x	= 0.0
.y	= 1.0

The Macro expansion would be as follows:

```
MULADD        EA = Rindex.xxxx,C0.xyxx,CstreamID;
MOV           EMx (x = 0 thru 4) = Rdata;
```

The SX will check for invalid writes and **mask out the data** so it won't be written to memory. Invalid writes are:

1) Index value >= Max Index value
2) bit 31 != 0 (negative index)
3) bits [30:23] != 23 + IEEE_EXP_BIAS (127) (meaning the index was too big to be represented using 23 bits)

They cannot have texture instructions interleaved in the export block. These exports **are not guaranteed to be ordered**.

Also, when doing a pass thru export**,** the shader must still do either a position and PC export (if Vertex) or a color export (if Pixel). The pass thru export can occur anywhere in any shader program and thus can be used to debug. There can be any number of pass thru export blocks throughout the pixel or vertex shader or both.

17.2 Arbitration restrictions

Here are the Sequencer arbitration restrictions:

1) Cannot execute a serialized thread if the corresponding texture pending bit and VC pending is set
2) Cannot allocate position if any older thread has not allocated position
3) Cannot execute a texture clause if texture reads are pending
4) Cannot execute a VC clause if VC reads are pending
5) Cannot execute last if texture pending (even if not serial)
6) Cannot allocate if not last for color exports.
7) Cannot allocate if not last for PC exports.

18. Export Types

The export type (or the location where the data should be put) is specified using the destination address field in the ALU instruction. Here is a list of all possible export modes:

18.1 Vertex Shading

0:15	- 16 parameter cache
16:31	- Empty (Reserved?)
32	- Export Address
33:37	- 5 vertex exports to the frame buffer and index
38:46	- Empty
47	- Debug Address
48:52	- 5 debug export (interpret as normal memory export)
53:59	- Empty
60	- export addressing mode
61	- Empty
62	- position
63	- sprite size export that goes with position export

(X= point size, Y= edge flag is bit 0, Z= VtxKill is bitwise OR of bits 30:0. Any bit other than sign means VtxKill.)

18.2 Pixel Shading

0	- Color for buffer 0 (primary)
1	- Color for buffer 1
2	- Color for buffer 2
3	- Color for buffer 3
4:15	- Empty
16	- Buffer 0 Color/Fog (primary)
17	- Buffer 1 Color/Fog
18	- Buffer 2 Color/Fog
19	- Buffer 3 Color/Fog
20:31	- Empty
32	- Export Address
33:37	- 5 exports for multipass pixel shaders.
38:46	- Empty
47	- Debug Address
48:52	- 5 debug exports (interpret as normal memory export)
60	- export addressing mode
61	- Z for primary buffer (Z exported to 'alpha' component)
62:63	- Empty

19. Special Interpolation modes

19.1 Real time commands

We are unable to use the parameter memory since there is no way for a command stream to write into it. Instead we need to add three 4x128 memories (one for each of three vertices x 4 interpolants). These will be mapped onto the register bus and written by type 0 packets, and output to the the parameter busses (the sequencer and/or PA need to be able to address the realtime parameter memory as well as the regular parameter store. This mode is triggered by the primitive type: REAL TIME. The actual memories are in the in the SX blocks. The parameter data memories are hooked on the RBBM bus and are loaded by the CP using register mapped memory.

19.2 Sprites/ XY screen coordinates/ FB information

XY screen coordinates may be needed in the shader program. This functionality is controlled by the param_gen register (in SQ) in conjunction with the SND_XY register (in SC) and the param_gen_pos. Also it is possible to send the faceness information (for OGL front/back special operations) to the shader using the same control register. Here is a list of all the modes and how they interact together:

The Data is going to be written in the register specified by the param_gen_pos register.

Param_Gen disable, snd_xy disable = No modification
Param_Gen disable, snd_xy enable = No modification
Param_Gen enable, snd_xy disable = Sign(faceness)garbage,(Sign Point)garbage,Sign(Line)s, t
Param_Gen enable, snd_xy enable = Sign(faceness)screenX,(Sign Point)screenY,Sign(Line)s, t

In other words,

The generated vector is (X in RED, Y in GREEN, S in BLUE and T in ALPHA):
X,Y,S,T

PGenReg.X = screen X biased 2^23 (assumes pixel center at 0.0), sign bit encodes faceness (0=frontface, 1=backface)
PGenReg.Y = screen Y biased 2^23 (assumes pixel center at 0.0), sign encodes is point primitive (0=not point, 1=is point)
PGenReg.Z = parametric S coordinate [0..1], sign encodes is line primitive (0=not line, 1=is line)

PGenReg.W = parametric T coordinate [0..1]

Constant
C0.X = 2^23 (debias for D3D)
C0.Y= 2^23 - 0.5 (debias for OGL which has pixel centers at 0.5)

To generate useable XY:
For D3D:
ADD ScreenXYReg.xy__ = abs(PGenReg), -C0.xxxx
For OGL
ADD ScreenXYReg.xy__ = abs(PGenReg), -C0.yyyy
Note abs has to be done on PGenReg

To access faceness.
Must ALWAYS use (or pos/neg test against) PGenReg.X.
< 0.0 is backface
>= 0.0 is frontface

To access parametric ST.
Same as before simply take abs before access.
realS = abs(PGenReg.Z)
realT = abs(PGenReg.W)

To access primitive type
+/-ZERO cannot be differentiated in shader pipe so a RECIP_CLAMPED instruction must be done first before testing isLine.
isPoint = PGenReg.Y (if <0.0 then point primitive)
isLine = RECIP_CLAMPED PGenReg.Z (if <0.0 then line primitive)
if ((isPoint>=0.0) && (isLine>=0.0)) then triangle primitive

19.3 Auto generated counters

In the cases we are dealing with multipass shaders, the sequencer is going to generate a vector count to be able to both use this count to write the 1st pass data to memory and then use the count to retrieve the data on the 2nd pass. The count is always generated in the same way but it is passed to the shader in a slightly different way depending on the shader type (pixel or vertex). This is toggled on and off using the GEN_INDEX_PIX/VTX register. The sequencer is going to keep two counters, one for pixels and one for vertices. Every time a full vector of vertices or pixels is written to the GPRs the counter is incremented. Every time a RST_PIX_COUNT or RST_VTX_COUNT events are received, the corresponding counter is reset. While there is only one count broadcast to the GPRs, the LSB are hardwired to specific values making the index different for all elements in the vector. Since the count must be different for all pixels/vertices and the 4 LSBs (16 positions) are hardwired to the corresponding shader unit the SQ has two choices:

1) Maintain a 17 bit counter that counts the vectors of 64. In this case the phase must be appended to the count before the count is broadcast to the SPs:

Counter (17 bits)	Phase (2 bits)	Hardwired (4 bits)

2) Maintain a 21 bits counter that counts sub-vectors of 16. In this case only the counter is sent to the Sps:

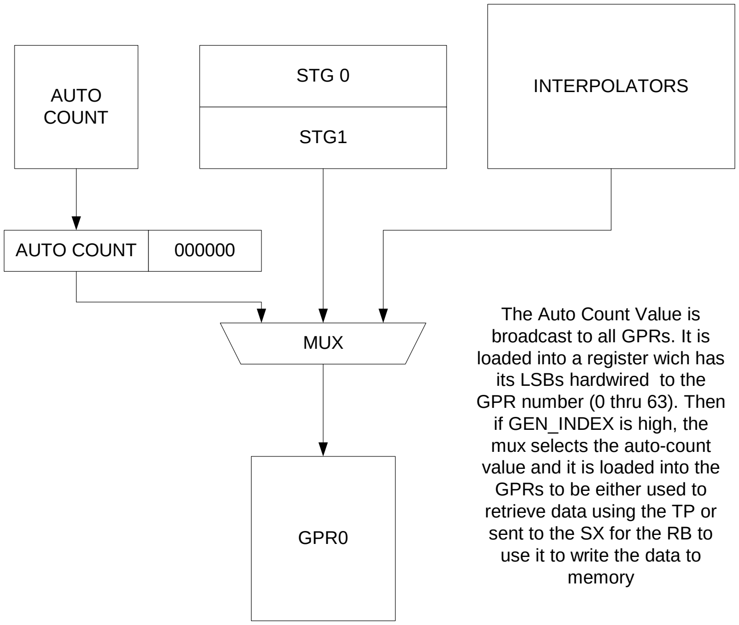
Counter (19 bits)	Hardwired (4 bits)

19.3.1 *Vertex shaders*

In the case of vertex shaders, if GEN_INDEX_VTX is set, the data will be put into the x field of the third register (it means that the compiler must allocate 3 GPRs in all multipass vertex shader modes).

19.3.2 *Pixel shaders*

In the case of pixel shaders, if GEN_INDEX_PIX is set, the data will be put in the x field of the param_gen_pos+1 register.



The Auto Count Value is broadcast to all GPRs. It is loaded into a register wich has its LSBs hardwired to the GPR number (0 thru 63). Then if GEN_INDEX is high, the mux selects the auto-count value and it is loaded into the GPRs to be either used to retrieve data using the TP or sent to the SX for the RB to use it to write the data to memory

Figure 10: GPR input mux Control

20. State management

Every clock, the sequencer will report to the CP the oldest states still in the pipe. These are the states of the programs as they enter the last ALU clause.

20.1 Parameter cache synchronization

In order for the sequencer not to begin a group of pixels before the associated group of vertices has finished, the sequencer will keep a 6 bit count per state (for a total of 8 counters). These counters are initialized to 0 and every time a vertex shader exports its data TO THE PARAMETER CACHE, the corresponding pointer is incremented. When the SC sends a new vector of pixels with the SC_SQ_new_vector bit asserted, the sequencer will first check if the count is greater than 0 before accepting the transmission (it will in fact accept the transmission but then lower its ready to receive). Then the sequencer waits for the count to go to one and decrements it. The sequencer can then issue the group of pixels to the interpolators. Every time the state changes, the new state counter is initialized to 0.

21. XY Address imports

The SC will be able to send the XY addresses to the GPRs. It does so by interleaving the writes of the IJs (to the IJ buffer) with XY writes (to the XY buffer). Then when writing the data to the GPRs, the sequencer is going to interpolate the IJ data or pass the XY data thru a Fix→float converter and expander and write the converted values to the GPRs. The Xys are currently SCREEN SPACE COORDINATES. The values in the XY buffers will wrap. See section 19.2 for details on how to control the interpolation in this mode.

21.1 Vertex indexes imports

In order to import vertex indexes, we have 16 8x96 staging registers. These are loaded one line at a time by the VGT block (96 bits). They are loaded in floating point format and can be transferred in 4 or 8 clocks to the GPRs.

22. Registers

Please see the auto-generated web pages for register definitions.

23. Interfaces

23.1 External Interfaces

Whenever an x is used, it means that the bus is broadcast to all units of the same name. For example, if a bus is named SQ→SPx it means that SQ is going to broadcast the same information to all SP instances.

23.2 SC to SP Interfaces

23.2.1 *SC_SP#*

There is one of these interfaces at front of each of the SP (buffer to stage pixel interpolators). This interface transmits the I,J data for pixel interpolation. For the entire system, two quads per clock are transferred to the 4 SPs, so each of these 4 interfaces transmits one half of a quad per clock. The interface below describes a half of a quad worth of data.
The actual data which is transferred per quad is
Ref Pix I => S4.20 Floating Point I value *4
Ref Pix J => S4.20 Floating Point J value *4

This equates to a total of 200 bits which transferred over 2 clocks
and therefor needs an interface 100 bits wide

Additionally, X,Y data (12-bit unsigned fixed) is conditionally sent across this data bus over the same wires in an additional clock. The X,Y data is sent on the lower 24 bits of the data bus with faceness in the msb.
Transfers across these interfaces are synchronized with the SC_SQ IJ Control Bus transfers.

The data transfer across each of these busses is controlled by a IJ_BUF_INUSE_COUNT in the SC. Each time the SC has sent a pixel vector's worth of data to the SPs, he will increment the IJ_BUF_INUSE_COUNT count. Prior to sending the next pixel vectors data, he will check to make sure the count is less than MAX_BUFER_MINUS_2, if not the SC will stall until the SQ returns a pipelined pulse to decrement the count when he has scheduled a buffer free. Note: We could/may optimize for the case of only sending only IJ to use all the buffers to pre-load more. Currently it is planned for the SP to hold 2 double buffers of I,J data and two buffers of X,Y data, so if either X,Y or Centers and Centroids are on, then the SC can send two Buffers.

In at least the initial version, the SC shall send 16 quads per pixel vector even if the vector is not full. This will increment buffer write address pointers correctly all the time. (We may revisit this for both the SX,SP,SQ and add a EndOfVector signal on all interfaces to quit early. We opted for the simple mode first with a belief that only the end of packet and multiple new vector signals should cause a partial vector and that this would not really be significant performance hit.)

Name	Bits	Description
SC_SP#_data	100	IJ information sent over 2 clocks (or X,Y in 24 LSBs with faceness in upper bit) **Type 0 or 1**, First clock I, second clk J Field ULC URC LLC LRC Bits [63:39] [38:26] [25:13] [12:0] Format SE4M20 SE4M20 SE4M20 SE4M20 **Type 2** Field Face X Y Bits [24] [23:12] [11:0] Format Bit Unsigned Unsigned
SC_SP#_valid	1	Valid
SC_SP#_last_quad_data	1	This bit will be set on the last transfer of data per quad.

SC_SP#_type	2	0 -> Indicates centroids 1 -> Indicates centers 2 -> Indicates X,Y Data and faceness on data bus The SC shall look at state data to determine how many types to send for the interpolation process.

The # is included for clarity in the spec and will be replaced with a prefix of u#_ in the verilog module statement for the SC and the SP block will have neither because the instantiation will insert the prefix.

23.2.2 SC_SQ

This is the control information sent to the sequencer in order to synchronize and control the interpolation and/or loading data into the GPRs needed to execute a shader program on the sent pixels. This data will be sent over two clocks per transfer with 1 to 16 transfers. Therefore the bus (approx 108 bits) could be folded in half to approx 54 bits.

Name	Bits	Description
SC_SQ_data	46	Control Data sent to the SQ 1 clk transfers Event – valid data consist of event_id and state_id. Instruct SQ to post an event vector to send state id and event_id through request fifo and onto the reservation stations making sure state id and/or event_id gets back to the CP. Events only follow end of packets so no pixel vectors will be in progress. Empty Quad Mask – Transfer Control data consisting of pc_dealloc or new_vector. Receipt of this is to transfer pc_dealloc or new_vector without any valid quad data. New vector will always be posted to request fifo and pc_dealloc will be attached to any pixel vector outstanding or posted in request fifo if no valid quad outstanding. 2 clk transfers Quad Data Valid – Sending quad data with or without new_vector or pc_dealloc. New vector will be posted to request fifo with or without a pixel vector and pc_dealloc will be posted with a pixel vector unless none is in progress. In this case the pc_dealloc will be posted in the request queue. Filler quads will be transferred with The Quad mask set but the pixel corresponding pixel mask set to zero.
SC_SQ_valid	1	SC sending valid data, 2nd clk could be all zeroes

SC_SQ_data – first clock and second clock transfers are shown in the table below.

Name	BitField	Bits	Description

1st Clock Transfer			
SC_SQ_event	0	1	This transfer is a 1 clock event vector Force quad_mask = new_vector=pc_dealloc=0
SC_SQ_event_id	[5:1]	4	This field identifies the event 0 => denotes an End Of State Event 1 => TBD
SC_SQ_state_id	[8:6]	3	State/constant pointer (6*3+3)
SC_SQ_pc_dealloc	[11:9]	3	Deallocation token for the Parameter Cache
SC_SQ_new_vector	12	1	The SQ must wait for Vertex shader done count > 0 and after dispatching the Pixel Vector the SQ will decrement the count.
SC_SQ_quad_mask	[16:13]	4	Quad Write mask left to right SP0 => SP3
SC_SQ_end_of_prim	17	1	End Of the primitive
SC_SQ_pix_mask	[33:18]	16	Valid bits for all pixels SP0=>SP3 (UL,UR,LL,LR)
SC_SQ_provok_vtx	[35:34]	2	Provoking vertex for flat shading
SC_SQ_lod_correct_0	[44:36]	9	LOD correction for quad 0 (SP0) (9 bits per quad)
SC_SQ_lod_correct_1	[53:45]	9	LOD correction for quad 1 (SP1) (9 bits per quad)
2nd Clock Transfer			
SC_SQ_lod_correct_2	[8:0]	9	LOD correction for quad 2 (SP2) (9 bits per quad)
SC_SQ_lod_correct_3	[17:9]	9	LOD correction for quad 3 (SP3) (9 bits per quad)
SC_SQ_pc_ptr0	[28:18]	11	Parameter Cache pointer for vertex 0
SC_SQ_pc_ptr1	[39:29]	11	Parameter Cache pointer for vertex 1
SC_SQ_pc_ptr2	[50:40]	11	Parameter Cache pointer for vertex 2
SC_SQ_prim_type	[53:51]	3	Stippled line and Real time command need to load tex cords from alternate buffer 000: Sprite (point) 001: Line 010: Tri_rect 100: Realtime Sprite (point) 101: Realtime Line 110: Realtime Tri_rect

Name	**Bits**	**Description**
SQ_SC_free_buff	1	Pipelined bit that instructs SC to decrement count of buffers in use.
SQ_SC_dec_cntr_cnt	1	Pipelined bit that instructs SC to decrement count of new vector and/or event sent to prevent SC from overflowing SQ interpolator/Reservation request fifo.

The scan converter will submit a partial vector whenever:

1.) He gets a primitive marked with an end of packet signal.

2.) A current pixel vector is being assembled with at least one or more valid quads and the vector has been marked for deallocate when a primitive marked new_vector arrives. The Scan Converter will submit a partial vector (up to 16quads with zero pixel mask to fill out the vector) prior to submitting the new_vector marker\primitive.

(This will prevent a hang which can be demonstrated when all primitives in a packet three vectors are culled except for a one quad primitive that gets marked pc_dealloc (vertices maximum size). In this case two new_vectors are submitted and processed, but then one valid quad with the pc_dealloc creates a vector and then the new would wait for another vertex vector to be processed, but the one being waited for could never export until the pc_dealloc signal made it through and thus the hang.)

23.2.3 *SQ to SX(SP): Interpolator bus*

Name	Direction	Bits	Description
SQ_SXx_interp_cyl_wrap	SQ→SXx	4	Which channel needs to be cylindrical wrapped
SQ_SXx_wrap_count	SQ->SXx	2	Cylindrical wrap count
SQ_SPx_auto_count	SQ->SPx	19	Auto generated count for VTx and Pixels
SQ_SPx_interp_param_gen	SQ→SPx	1	Generate Parameter
SQ_SPx_interp_prim_type	SQ→SPx	2	Bits [1:0] of primitive type sent by SC
SQ_SPx_interp_buff_swap	SQ→SPx	1	Swap IJ buffers
SQ_SPx_interp_IJ_line	SQ→SPx	2	IJ line number
SQ_SPx_interp_mode	SQ→SPx	1	Center/Centroid sampling
SQ_SXx_pc_ptr0	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_pc_ptr1	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_pc_ptr2	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_rt_sel	SQ→SXx	1	Selects between RT and Normal data (Bit 2 of prim type)
SQ_SX0_pc_wr_en	SQ→SX0	8	Write enable for the PC memories
SQ_SX1_pc_wr_en	SQ→SX1	8	Write enable for the PC memories
SQ_SXx_pc_wr_addr	SQ→SXx	7	Write address for the PCs
SQ_SXx_pc_channel_mask	SQ→SXx	4	Channel mask
SQ_SXx_pc_ptr_valid	SQ→SXx	1	Read pointers are valid.
SQ_SPx_interp_valid	SQ→SPx	1	Interpolation control valid

23.2.4 *SQ to SP: Staging Register Data*

This is a broadcast bus that sends the VSISR information to the staging registers of the shader pipes.

Name	Direction	Bits	Description
SQ_SPx_vsr_data	SQ→SPx	96	Pointers of indexes or HOS surface information
SQ_SPx_vsr_wrt_addr	SQ→SPx	3	Staging register write address
SQ_SPx_vsr_rd_addr	SQ→SPx	3	Staging register read address
SQ_SP0_ vsr_valid	SQ→SP0	1	Data is valid
SQ_SP1_ vsr_ valid	SQ→SP1	1	Data is valid
SQ_SP2_ vsr_ valid	SQ→SP2	1	Data is valid
SQ_SP3_ vsr_ valid	SQ→SP3	1	Data is valid
SQ_SPx_vsr_read	SQ→SPx	1	Increment the read pointers

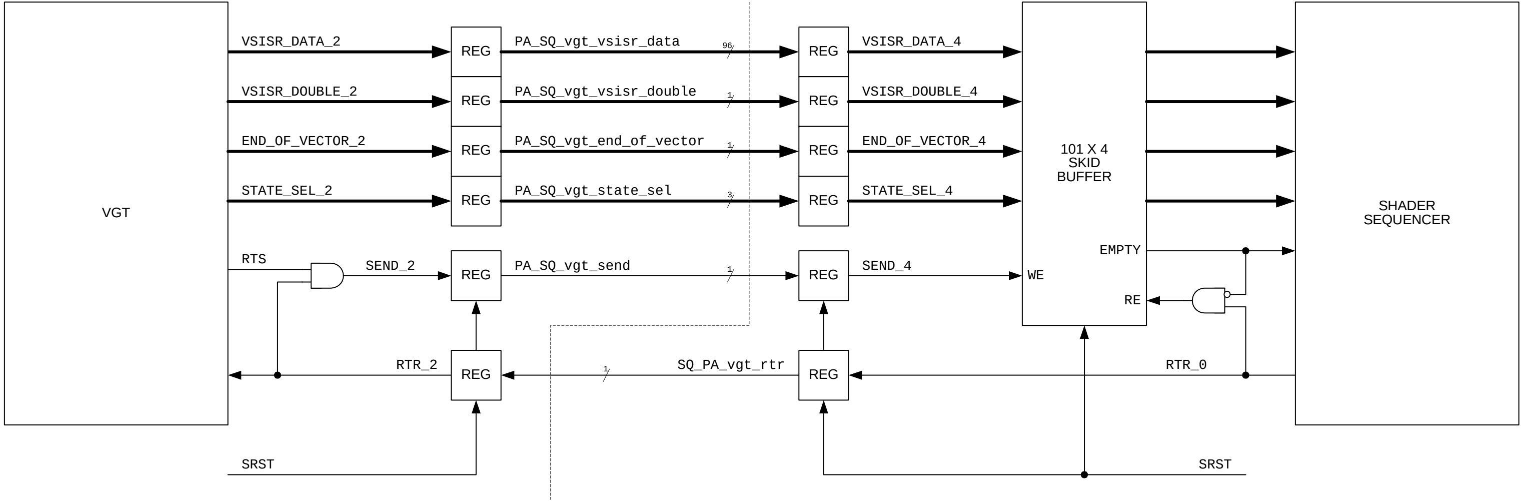
23.2.5 *VGT to SQ : Vertex interface*

23.2.5.1 Interface Signal Table

The area difference between the two methods is not sufficient to warrant complicating the interface or the state requirements of the VSISRs. **Therefore, the POR for this interface is that the VGT will transmit the data to the VSISRs (via the Shader Sequencer) in full, 32-bit floating-point format.** The VGT can transmit up to six 32-bit floating-point values to each VSISR where four or more values require two transmission clocks. The data bus is 96 bits wide. In the case where an event is sent the 5 LSBs of VGT_SQ_vsisr_data contain the eventID.

Name	Bits	Description
VGT_SQ_vsisr_data	96	Pointers of indexes or HOS surface information
VGT_SQ_event	1	VGT is sending an event
VGT_SQ_vsisr_continued	1	0: Normal 96 bits per vert 1: double 192 bits per vert
VGT_SQ_end_of_vtx_vect	1	Indicates the last VSISR data set for the current process vector (for double vector data, "end_of_vector" is set on the first vector)
VGT_SQ_indx_valid	1	Vsisr data is valid
VGT_SQ_state	3	Render State (6*3+3 for constants). This signal is guaranteed to be correct when “VGT_SQ_vgt_end_of_vector” is high.
VGT_SQ_send	1	Data on the VGT_SQ is valid receive (see write-up for standard R400 SEND/RTR interface handshaking)
SQ_VGT_rtr	1	Ready to receive (see write-up for standard R400 SEND/RTR interface handshaking)

23.2.5.2 Interface Diagrams



VGT
VSISR_DATA_2
VSISR_DOUBLE_2
END_OF_VECTOR_2
STATE_SEL_2
RTS
SEND_2
REG
PA_SQ_vgt_vsisr_data
96
PA_SQ_vgt_vsisr_double
1
PA_SQ_vgt_end_of_vector
1
PA_SQ_vgt_state_sel
3
PA_SQ_vgt_send
1
VSISR_DATA_4
VSISR_DOUBLE_4
END_OF_VECTOR_4
STATE_SEL_4
SEND_4
101 X 4
SKID
BUFFER
EMPTY
WE
RE
SHADER
SEQUENCER
RTR_2
1
SQ_PA_vgt_rtr
RTR_0
SRST
SRST

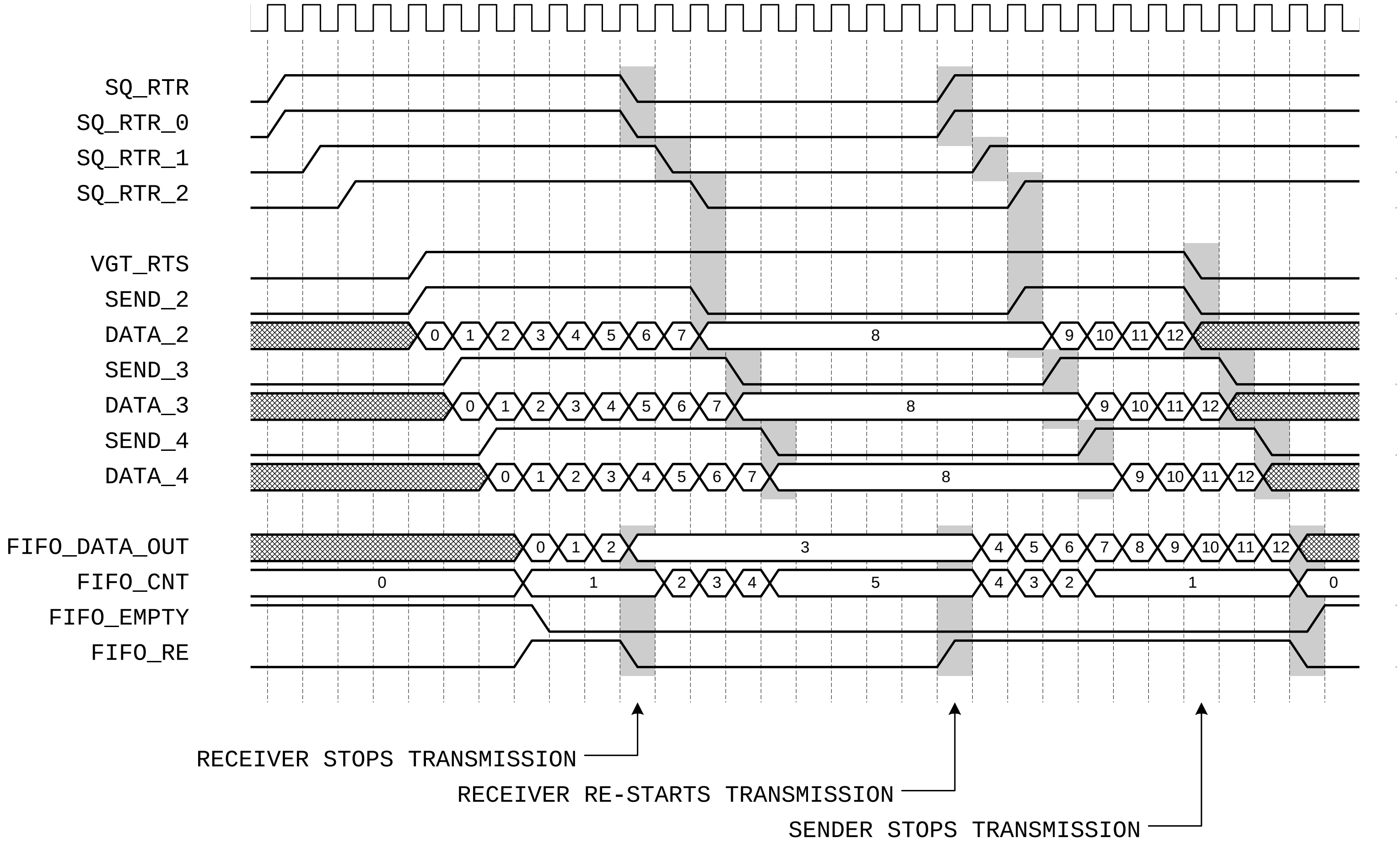


Figure 1. Detailed Logical Diagram for PA_SQ_vgt Interface.

23.2.6 *SQ to SX: Control bus*

Name	Direction	Bits	Description
SQ_SXx_exp_type	SQ→SXx	2	00: Pixel without z (1 to 4 buffers) 01: Pixel with z (1 to 4 buffers) 10: Position (1 or 2 results) 11: Pass thru (1 to 5 results aligned)
SQ_SXx_exp_number	SQ→SXx	2	Number of locations needed in the export buffer (encoding depends on the type see bellow).
SQ_SXx_exp_alu_id	SQ→SXx	4	ALU ID. Revolving ID 0 thru 15. Memory exports have to increment this count by 4 or 8 depending on the size requested. Other type of exports increment the ID by 1.
SQ_SXx_exp_valid	SQ→SXx	1	Valid bit
SQ_SXx_exp_state	SQ→SXx	3	State Context
SQ_SXx_free_done	SQ→SXx	1	Pulse that indicates that the previous export is finished **from the point of view of the SP. This does not necessarily mean that the data has been transferred to RB or PA, or that the space in export buffer for that particular vector thread has been freed up.**
SQ_SXx_free_alu_id	SQ→SXx	4	ALU ID that was used at allocate time.

Depending on the type the number of export location changes:

- Type 00 : Pixels without Z
  - 00 = 1 buffer
  - 01 = 2 buffers
  - 10 = 3 buffers
  - 11 = 4 buffer
- Type 01: Pixels with Z
  - 00 = 2 Buffers (color + Z)
  - 01 = 3 buffers (2 color + Z)
  - 10 = 4 buffers (3 color + Z)
  - 11 = 5 buffers (4 color + Z)
- Type 10 : Position export
  - 00 = 1 position
  - 01 = 2 positions
  - 1X = Undefined
- Type 11: Pass Thru
  - 00 = 4 buffers
  - 01 = 8 buffers
  - 10 = Undefined
  - 11 = Undefined

Below the thick black line is the end of transfer packet that tells the SX that a given export is finished. The report packet **will always arrive either before or at the same time than the next export to the same ALU id**.

23.2.7 *SX to SQ : Output file control*

Name	Direction	Bits	Description
SXx_SQ_pix_free_count0	SXx→SQ	6	How many slots where just freed in the SX for bank0
SXx_SQ_pix_count0_valid	SXx→SQ	1	Free_count0 is valid
SXx_SQ_pix_free_count1	SXx→SQ	6	How many slots where just freed in the SX for bank1
SXx_SQ_pix_count1_valid	SXx→SQ	1	Free_count1 is valid
SXx_SQ_pos_free_count0	SXx→SQ	4	How many slots where just freed in the SX for bank0
SXx_SQ_pos_count0_valid	SXx→SQ	1	Free_count0 is valid
SXx_SQ_pos_free_count1	SXx→SQ	4	How many slots where just freed in the SX for bank1

SXx_SQ_pos_count1_valid	SXx→SQ	1	Free_count1 is valid
SXx_SQ_mem_export_free	SXx→SQ	1	Freed a memory export slot

23.2.8 *SQ to TP: Control bus*

Once every clock, the fetch unit sends to the sequencer on which RS line it is now working and if the data in the GPRs is ready or not. This way the sequencer can update the fetch valid bits flags for the reservation station. The sequencer also provides the instruction and constants for the fetch to execute and the address in the register file where to write the fetch return data.

Name	Direction	Bits	Description
TPx_SQ_data_rdy	TPx→ SQ	1	Data ready
TPx_SQ_rs_line_num	TPx→ SQ	6	Line number in the Reservation station
TPx_SQ_type	TPx→ SQ	1	Type of data sent (0:PIXEL, 1:VERTEX)
SQ_TPx_send	SQ→TPx	1	Sending valid data
SQ_TPx_const	SQ→TPx	48	Fetch state sent over 4 clocks (192 bits total)
SQ_TPx_instr	SQ→TPx	24	Fetch instruction sent over 4 clocks
SQ_TPx_end_of_group	SQ→TPx	1	Last instruction of the group
SQ_TPx_Type	SQ→TPx	1	Type of data sent (0:PIXEL, 1:VERTEX)
SQ_TPx_gpr_phase	SQ→TPx	2	Write phase signal
SQ_TP0_lod_correct	SQ→TP0	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP0_pix_mask	SQ→TP0	4	Pixel mask 1 bit per pixel
SQ_TP1_lod_correct	SQ→TP1	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP1_pix_mask	SQ→TP1	4	Pixel mask 1 bit per pixel
SQ_TP2_lod_correct	SQ→TP2	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP2_pix_mask	SQ→TP2	4	Pixel mask 1 bit per pixel
SQ_TP3_lod_correct	SQ→TP3	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP3_pix_mask	SQ→TP3	4	Pixel mask 1 bit per pixel
SQ_TPx_rs_line_num	SQ→TPx	6	Line number in the Reservation station
SQ_TPx_write_gpr_index	SQ->TPx	7	Index into Register file for write of returned Fetch Data
SQ_TPx_ctx_id	SQ→TPx	3	The state context ID (needed for multisample resolves)
SQ_TPx_SIMD	SQ->TPx	1	Tells the TP from which SIMD the data is coming from.

23.2.9 *SQ to VC: Control bus*

Once every clock, the VC unit sends to the sequencer on which RS line it is now working and if the data in the GPRs is ready or not. This way the sequencer can update the fetch valid bits flags for the reservation station. The sequencer also provides the instruction and constants for the fetch to execute and the address in the register file where to write the fetch return data.

Name	Direction	Bits	Description
VCx_SQ_data_rdy	VCx→ SQ	1	Data ready
VCx_SQ_rs_line_num	VCx→ SQ	6	Line number in the Reservation station
VCx_SQ_type	VCx→ SQ	1	Type of data sent (0:PIXEL, 1:VERTEX)
SQ_VCx_send	SQ→VCx	1	Sending valid data
SQ_VCx_const	SQ→VCx	48	Fetch state sent over 4 clocks (192 bits total)
SQ_VCx_instr	SQ→VCx	24	Fetch instruction sent over 4 clocks
SQ_VCx_end_of_group	SQ→VCx	1	Last instruction of the group
SQ_VCx_Type	SQ→VCx	1	Type of data sent (0:PIXEL, 1:VERTEX)
SQ_VCx_gpr_phase	SQ→VCx	2	Write phase signal
SQ_VC0_pix_mask	SQ→VC0	4	Pixel mask 1 bit per pixel
SQ_VC1_pix_mask	SQ→VC1	4	Pixel mask 1 bit per pixel
SQ_VC2_pix_mask	SQ→VC2	4	Pixel mask 1 bit per pixel
SQ_VC3_pix_mask	SQ→VC3	4	Pixel mask 1 bit per pixel
SQ_VCx_rs_line_num	SQ→VCx	6	Line number in the Reservation station

SQ_VCx_write_gpr_index	SQ->VCx	7	Index into Register file for write of returned Fetch Data
SQ_VCx_SIMD	SQ->VCx	1	Tells the VC from which SIMD the data is coming from.

23.2.10 *TP to SQ: Texture stall*

The TP sends this signal to the SQ and the SPs when frees up a buffer.

Name	**Direction**	**Bits**	**Description**
TP_SQ_fetch_dec	TP→ SQ	1	Just freed a slot in the TP.

23.2.11 *VC to SQ: Vertex Cache stall*

The VC sends this signal to the SQ and the SPs when frees up a buffer. There are 2 types of buffers, Mega and Mini and a signal for both.

Name	**Direction**	**Bits**	**Description**
VC_SQ_fetch_dec_mega	VC→ SQ	1	Freed a Mega slot in the VC.
VC_SQ_fetch_dec_mini	VC→ SQ	1	Freed a Mini slot in the VC.

23.2.12 *SQ to SP: GPR and auto counter*

Name	**Direction**	**Bits**	**Description**
SQ_SPx_simd0_gpr_wr_addr	SQ→SPx	7	Write address
SQ_SPx_simd0_gpr_rd_addr	SQ→SPx	7	Read address
SQ_SPx_simd0_gpr_rd_en	SQ→SPx	1	Read Enable
SQ_SP0_simd0_gpr_pspv_wr_en	SQ→SP0 (SP1)	4	Write Enable for the GPRs of SP0-1 for PS and PV
SQ_SP2_simd0_gpr_pspv_wr_en	SQ→SP2 (SP3)	4	Write Enable for the GPRs of SP2-3 for PS and PV
SQ_SP4_simd0_gpr_pspv_wr_en	SQ→SP4 (SP5)	4	Write Enable for the GPRs of SP4-5 for PS and PV
SQ_SP6_simd0_gpr_pspv_wr_en	SQ→SP6 (SP7)	4	Write Enable for the GPRs of SP6-7 for PS and PV
SQ_SP0_simd0_gpr_int_wr_en	SQ→SP0	1	Write Enable for the GPRs of SP0 for Inputs (interp/vtx)
SQ_SP2_simd0_gpr_int_wr_en	SQ→SP2	1	Write Enable for the GPRs of SP1 for Inputs (interp/vtx)
SQ_SP4_simd0_gpr_int_wr_en	SQ→SP4	1	Write Enable for the GPRs of SP2 for Inputs (interp/vtx)
SQ_SP6_simd0_gpr_int_wr_en	SQ→SP6	1	Write Enable for the GPRs of SP3 for Inputs (interp/vtx)
SQ_SPx_gpr_phase	SQ→SPx	2	The phase mux (arbitrates between inputs, ALU SRC reads and writes)
SQ_SPx_simd0_channel_mask	SQ→SPx	4	The channel mask for SIMD0
SQ_SPx_gpr_input_sel	SQ→SPx	2	When the phase mux selects the inputs this tells from which source to read from: Interpolated data, VSR, autogen counter.
SQ_SPx_auto_count	SQ→SPx	21	Auto count generated by the SQ, common for all shader pipes
SQ_SPx_simd0_fetch_swizzle	SQ→SPx	6	Swizzle code for the TP request (2 bits per channel ignore W as it is not used). Bits [1..0] X mode select: 0=GPR_X 1=GPR_Y 2=GPR_Z 3=GPR_W Bits [3..2] Y mode select: 0=GPR_X 1=GPR_Y 2=GPR_Z 3=GPR_W Bits [5..4] Z mode select: 0=GPR_X 1=GPR_Y 2=GPR_Z 3=GPR_W
SQ_SPx_tp_fetch_simd_sel	SQ→SPx	1	TP Resource coming from: 0: SIMD0 1: SIMD1
SQ_SPx_vc_fetch_simd_sel	SQ→SPx	1	VC Resource coming from: 0: SIMD0 1: SIMD1
SQ_SPx_simd1_gpr_wr_addr	SQ→SPx	7	Write address

SQ_SPx_simd1_gpr_rd_addr	SQ→SPx	7	Read address
SQ_SPx_simd1_gpr_rd_en	SQ→SPx	1	Read Enable
SQ_SP0_simd1_gpr_pspv_wr_en	SQ→SP0 (SP1)	4	Write Enable for the GPRs of SP0-1 for PS and PV
SQ_SP2_simd1_gpr_pspv_wr_en	SQ→SP2 (SP3)	4	Write Enable for the GPRs of SP2-3 for PS and PV
SQ_SP4_simd1_gpr_pspv_wr_en	SQ→SP4 (SP5)	4	Write Enable for the GPRs of SP4-5 for PS and PV
SQ_SP6_simd1_gpr_pspv_wr_en	SQ→SP6 (SP7)	4	Write Enable for the GPRs of SP6-7 for PS and PV
SQ_SPx__simd1_channel_mask	SQ→SPx	4	The channel mask for SIMD1
SQ_SPx_simd1_fetch_swizzle	SQ→SPx	6	Swizzle code for the TP request (2 bits per channel ignore W as it is not used). Bits [1..0] X mode select: 0=GPR_X 1=GPR_Y 2=GPR_Z 3=GPR_W Bits [3..2] Y mode select: 0=GPR_X 1=GPR_Y 2=GPR_Z 3=GPR_W Bits [5..4] Z mode select: 0=GPR_X 1=GPR_Y 2=GPR_Z 3=GPR_W
SQ_SP0_simd1_gpr_int_wr_en	SQ→SP0	1	Write Enable for the GPRs of SP0-1 for Inputs (interp/vtx)
SQ_SP2_simd1_gpr_int_wr_en	SQ→SP2	1	Write Enable for the GPRs of SP2-3 for Inputs (interp/vtx)
SQ_SP4_simd1_gpr_int_wr_en	SQ→SP4	1	Write Enable for the GPRs of SP4-5 for Inputs (interp/vtx)
SQ_SP6_simd1_gpr_int_wr_en	SQ→SP6	1	Write Enable for the GPRs of SP6-7 for Inputs (interp/vtx)

23.2.13 *SQ to SPx:*

Name	Direction	Bits	Description
SQ_SPx_instr_start	SQ→SPx	1	Instruction start
SQ_SPx_simd0_instruct	SQ→SPx	24	Transferred over 4 cycles 0: SRC A Negate Argument Modifier 0:0 SRC A Abs Argument Modifier 1:1 SRC A Swizzle 9:2 Vector Dst 15:10 Per channel Select 23:16 00: GPR 01: PV 10: PS 11: Constant (if 11 has to be 11 for all channels) --- - 1: SRC B Negate Argument Modifier 0:0 SRC B Abs Argument Modifier 1:1 SRC B Swizzle 9:2 Scalar Dst 15:10 Per channel Select 23:16 00: GPR 01: PV 10: PS 11: Constant (if 11 has to be 11 for all channels) --- - 2: SRC C Negate Argument Modifier 0:0 SRC C Abs Argument Modifier 1:1 SRC C Swizzle 9:2 Unused 15:10 Per channel Select 23:16 00: GPR 01: PV 10: PS 11: Constant (if 11 has to be 11 for all channels) --- - 3: Vector Opcode 4:0 Scalar Opcode 10:5 Vector Clamp 11:11 Scalar Clamp 12:12 Vector Write Mask 16:13 Scalar Write Mask 20:17 Unused 23:21
SQ_SP0_simd0_pred_override	SQ→SP0 (SP1)	4	SP0 receives 2 bits and SP1 two bits as well. 0: Use per channel RGBA field (enables the per channel logic). 1: Use GPR for PV or PS settings. LET the 11 (constant) go thru unchanged
SQ_SP2_simd0_pred_override	SQ→SP2 (SP3)	4	0: Use per channel RGBA field (enables the per channel logic). 1: Use GPR for PV or PS settings. LET the 11 (constant) go thru unchanged
SQ_SP4_simd0_pred_override	SQ→SP4 (SP5)	4	0: Use per channel RGBA field (enables the per channel logic). 1: Use GPR for PV or PS settings. LET the 11 (constant) go thru unchanged

SQ_SP6_simd0_pred_override	SQ→SP6 (SP7)	4	0: Use per channel RGBA field (enables the per channel logic). 1: Use GPR for PV or PS settings. LET the 11 (constant) go thru unchanged
SQ_SPx_simd0_stall	SQ→SPx	1	Stall signal
SQ_SPx_simd1_instruct	SQ→SPx	24	Transferred over 4 cycles 0: SRC A Negate Argument Modifier 0:0 SRC A Abs Argument Modifier 1:1 SRC A Swizzle 9:2 Vector Dst 15:10 Per channel Select 23:16 00: GPR 01: PV 10: PS 11: Constant (if 11 has to be 11 for all channels) --- - 1: SRC B Negate Argument Modifier 0:0 SRC B Abs Argument Modifier 1:1 SRC B Swizzle 9:2 Scalar Dst 15:10 Per channel Select 23:16 00: GPR 01: PV 10: PS 11: Constant (if 11 has to be 11 for all channels) --- - 2: SRC C Negate Argument Modifier 0:0 SRC C Abs Argument Modifier 1:1 SRC C Swizzle 9:2 Unused 15:10 Per channel Select 23:16 00: GPR 01: PV 10: PS 11: Constant (if 11 has to be 11 for all channels) --- - 3: Vector Opcode 4:0 Scalar Opcode 10:5 Vector Clamp 11:11 Scalar Clamp 12:12 Vector Write Mask 16:13 Scalar Write Mask 20:17 Unused 23:21
SQ_SP0_simd0_pred_override	SQ→SP0 (SP1)	4	SP0 receives 2 bits and SP1 two bits as well. 0: Use per channel RGBA field (enables the per channel logic). 1: Use GPR for PV or PS settings. LET the 11 (constant) go thru unchanged
SQ_SP2_simd0_pred_override	SQ→SP2 (SP3)	4	0: Use per channel RGBA field (enables the per channel logic). 1: Use GPR for PV or PS settings. LET the 11 (constant) go thru unchanged
SQ_SP4_simd0_pred_override	SQ→SP4 (SP5)	4	0: Use per channel RGBA field (enables the per channel logic). 1: Use GPR for PV or PS settings. LET the 11 (constant) go thru unchanged

SQ_SP6_simd0_pred_override	SQ→SP6 (SP7)	4	0: Use per channel RGBA field (enables the per channel logic). 1: Use GPR for PV or PS settings. LET the 11 (constant) go thru unchanged
SQ_SPx_simd1_stall	SQ→SPx	1	Stall signal
SQ_SPx_export_simd_sel	SQ->SPx	1	Which SIMD engine is exporting.

23.2.14 SQ to SX: write mask interface (must be aligned with the SP data)

Name	Direction	Bits	Description
SQ_SX0_write_mask	SQ→SX0	8	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock. This is for the data coming of SP0 and SP2.
SQ_SX1_ write_mask	SQ→SX1	8	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock. This is for the data coming of SP1 and SP3.
SQ_SXx_channel_mask	SQ->SXx	4	This is the per channel export mask. It is computed by doing vector_mask | scalar_mask | bit 14 of the alu instruction.
SQ_SX0_kill_mask	SQ->SX0	8	These are the valid bits coming straight from the reservation stations.
SQ_SX1_kill_mask	SQ->SX1	8	These are the valid bits coming straight from the reservation stations.

23.2.15 *SP to SQ: Constant address load/ Predicate Set/Kill set*

Name	Direction	Bits	Description
SP0_SQ_simd0_const_addr	(SP1) SP0→SQ	36	Constant address load 18 bits from SP0 and 18 from SP4.
SP0_SQ_simd0_valid	SP0→SQ	1	Data valid
SP2_SQ_simd0_const_addr	(SP3) SP2→SQ	36	Constant address load
SP2_SQ_simd0_valid	SP2→SQ	1	Data valid
SP4_SQ_simd0_const_addr	(SP5) SP4→SQ	36	Constant address load
SP4_SQ_simd0_valid	SP4→SQ	1	Data valid
SP6_SQ_simd0_const_addr	(SP7) SP6→SQ	36	Constant address load
SP6_SQ_simd0_valid	SP6→SQ	1	Data valid
SP0_SQ_simd0_pred_kill_vector	(SP1) SP0→SQ	4	Data (predicates or kill/mask) 2 bits from SP0 and 2 bits from SP4
SP0_SQ_simd0_pred_kill_valid	SP0->SQ	1	Data valid
SP0_SQ_simd0_pred_kill_type	SP0->SQ	1	0: predicate vector 1: kill/mask vector
SP2_SQ_simd0_pred_kill_vector	(SP3) SP2→SQ	4	Data (predicates or kill/mask)
SP2_SQ_simd0_pred_kill_valid	SP2->SQ	1	Data valid
SP2_SQ_simd0_pred_kill_type	SP2->SQ	1	0: predicate vector 1: kill/mask vector
SP4_SQ_simd0_pred_kill_vector	(SP5) SP4→SQ	4	Data (predicates or kill/mask)
SP4_SQ_simd0_pred_kill_valid	SP4->SQ	1	Data valid
SP4_SQ_simd0_pred_kill_type	SP4->SQ	1	0: predicate vector 1: kill/mask vector
SP6_SQ_simd0_pred_kill_vector	(SP7) SP6→SQ	4	Data (predicates or kill/mask)
SP6_SQ_simd0_pred_kill_valid	SP6->SQ	1	Data valid
SP6_SQ_simd0_pred_kill_type	SP6->SQ	1	0: predicate vector 1: kill/mask vector
SP0_SQ_simd1_const_addr	(SP1) SP0→SQ	36	Constant address load 18 bits from SP0 and 18 from SP4.
SP0_SQ_simd1_valid	SP0→SQ	1	Data valid
SP2_SQ_simd1_const_addr	(SP3) SP2→SQ	36	Constant address load
SP2_SQ_simd1_valid	SP2→SQ	1	Data valid
SP4_SQ_simd1_const_addr	(SP5) SP4→SQ	36	Constant address load
SP4_SQ_simd1_valid	SP4→SQ	1	Data valid
SP6_SQ_simd1_const_addr	(SP7) SP6→SQ	36	Constant address load
SP6_SQ_simd1_valid	SP6→SQ	1	Data valid
SP0_SQ_simd1_pred_kill_vector	(SP1) SP0→SQ	4	Data (predicates or kill/mask) 2 bits from SP0 and 2 bits from SP4
SP0_SQ_simd1_pred_kill_valid	SP0->SQ	1	Data valid
SP0_SQ_simd1_pred_kill_type	SP0->SQ	1	0: predicate vector 1: kill/mask vector
SP2_SQ_simd1_pred_kill_vector	(SP3) SP2→SQ	4	Data (predicates or kill/mask)
SP2_SQ_simd1_pred_kill_valid	SP2->SQ	1	Data valid
SP2_SQ_simd1_pred_kill_type	SP2->SQ	1	0: predicate vector 1: kill/mask vector
SP4_SQ_simd1_pred_kill_vector	(SP5) SP4→SQ	4	Data (predicates or kill/mask)
SP4_SQ_simd1_pred_kill_valid	SP4->SQ	1	Data valid
SP4_SQ_simd1_pred_kill_type	SP4->SQ	1	0: predicate vector 1: kill/mask vector
SP6_SQ_simd1_pred_kill_vector	(SP7) SP6→SQ	4	Data (predicates or kill/mask)
SP6_SQ_simd1_pred_kill_valid	SP6->SQ	1	Data valid
SP6_SQ_simd1_pred_kill_type	SP6->SQ	1	0: predicate vector 1: kill/mask vector

Because of the sharing of the bus none of the MOVA, PREDSET or KILL instructions may be coissued.

23.2.16 *SQ to SPx: constant broadcast*

Name	Direction	Bits	Description

SQ_SPx_simd0_const	SQ→SPx	128	Constant broadcast
SQ_SPx_simd1_const	SQ→SPx	128	Constant broadcast
SQ_SPx_simd0_const_sel	SQ→SPx	2	Use the incoming constant instead of the registered one for the next group of 16. 0 : Normal mode 1: Waterfall on SRCA 2: Waterfall on SRCB 3: Waterfall on SRCC
SQ_SPx_simd1_const_sel	SQ→SPx	2	Use the incoming constant instead of the registered one for the next group of 16. 0 : Normal mode 1: Waterfall on SRCA 2: Waterfall on SRCB 3: Waterfall on SRCC

23.2.17 *SQ to CP: RBBM bus*

Name	Direction	Bits	Description
SQ_RBB_rs	SQ→CP	1	Read Strobe
SQ_RBB_rd	SQ→CP	32	Read Data
SQ_RBBM_nrtrtr	SQ→CP	1	Optional
SQ_RBBM_rtr	SQ→CP	1	Real-Time (Optional)

23.2.18 *CP to SQ: RBBM bus*

Name	Direction	Bits	Description
rbbm_we	CP→SQ	1	Write Enable
rbbm_a	CP→SQ	15	Address -- Upper Extent is TBD (16:2)
rbbm_wd	CP→SQ	32	Data
rbbm_be	CP→SQ	4	Byte Enables
rbbm_re	CP→SQ	1	Read Enable
rbb_rs0	CP→SQ	1	Read Return Strobe 0
rbb_rs1	CP→SQ	1	Read Return Strobe 1
rbb_rd0	CP→SQ	32	Read Data 0
rbb_rd1	CP→SQ	32	Read Data 0
RBBM_SQ_soft_reset	CP→SQ	1	Soft Reset

23.2.19 *SQ to CP: State report*

Name	Direction	Bits	Description
SQ_CP_vs_event	SQ→CP	1	Vertex Shader Event
SQ_CP_vs_eventid	SQ→CP	5	Vertex Shader Event ID
SQ_CP_ps_event	SQ→CP	1	Pixel Shader Event
SQ_CP_ps_eventid	SQ→CP	5	Pixel Shader Event ID

23.3 Example of control flow program execution

We now provide some examples of execution to better illustrate the new design.

Given the program:

Alu 0
Alu 1
Tex 0
Tex 1
Alu 3 Serial
Alu 4
Tex 2
Alu 5
Alu 6 Serial

Tex 3
Alu 7
Alloc Position 1 buffer
Alu 8 Export
Tex 4
Alloc Parameter 3 buffers
Alu 9 Export 0
Tex 5
Alu 10 Serial Export 2
Alu 11 Export 1 End

Would be converted into the following CF instructions:

```
Execute 0 Alu 0 Alu 0 Tex 0 Tex 1 Alu 0 Alu 0 Tex 0 Alu 1 Alu 0 Tex
Execute 0 Alu
Alloc Position 1
Execute 0 Alu 0 Tex
Alloc Param 3
Execute_end 0 Alu 0 Tex 1 Alu 0 Alu
```

And the execution of this program would look like this:

Put thread in Vertex RS:

Control Flow Instruction Pointer (12 bits), (CFP)
Execution Count Marker (3 or 4 bits), (ECM)
Loop Iterators (4x9 bits), (LI)
Call return pointers (4x12 bits), (CRP)
Predicate Bits(4x64 bits), (PB)
Export ID (1 bit), (EXID)
GPR Base Ptr (8 bits), (GPR)
Export Base Ptr (7 bits), (EB)
Context Ptr (3 bits).(CPTR)
LOD correction bits (16x6 bits) (LOD)

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	0	0	0	0	0	0	0	0	0

Valid Thread (VALID)
Texture/ALU engine needed (TYPE)
Texture Reads are outstanding (PENDING)
Waiting on Texture Read to Complete (SERIAL)
Allocation Wait (2 bits) (ALLOC)
    00 – No allocation needed
    01 – Position export allocation needed (ordered export)
    10 – Parameter or pixel export needed (ordered export)
    11 – pass thru (out of order export)
Allocation Size (4 bits) (SIZE)
Position Allocated (POS_ALLOC)
First thread of a new context (FIRST)
Last (1 bit), (LAST)

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	0	0	0	0	0	1	0

Then the thread is picked up for the execution of the first control flow instruction:

```
Execute 0 Alu 0 Alu 0 Tex 0 Tex 1 Alu 0 Alu 0 Tex 0 Alu 1 Alu 0 Tex
```

It executes the first two ALU instructions and goes back to the RS for a resource request change. Here is the state returned to the RS:

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	2	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	0	0	0	0	0	1	0

Then when the texture pipe frees up, the arbiter picks up the thread to issue the texture reads. The thread comes back in this state:

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	4	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	1	0	0	0	1	0

Because of the serial bit the arbiter must wait for the texture to return and clear the PENDING bit before it can pick the thread up. Lets say that the texture reads are complete, then the arbiter picks up the thread and returns it in this state:

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	6	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	0	0	0	0	0	1	0

Again the TP frees up, the arbiter picks up the thread and executes. It returns in this state:

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	7	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	0	0	0	1	0

Now, even if the texture has not returned we can still pick up the thread for ALU execution because the serial bit is not set. The thread will however come back to the RS for the second ALU instruction because it has the serial bit set.

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	8	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	1	0	0	0	1	0

As soon as the TP clears the pending bit the thread is picked up and returns:

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
0	9	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	0	0	0	0	0	1	0

Picked up by the TP and returns:

```
Execute 0 Alu
```

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
1	0	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	0	0	0	1	0

Picked up by the ALU and returns (lets say the TP has not returned yet):

```
Alloc Position 1
```

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
2	0	0	0	0	0	0	0	0	0

Status Bits								
VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	01	1	0	1	0

If the SX has the place for the export, the SQ is going to allocate and pick up the thread for execution. It returns to the RS in this state:

```
Execute 0 Alu 0 Tex
```

State Bits									
CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
3	1	0	0	0	0	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	1	0	0	0	1	1	0

Now, since the TP has not returned yet, we must wait for it to return because we cannot issue multiple texture requests. The TP returns, clears the PENDING bit and we proceed:

```
Alloc Param 3
```

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
4	0	0	0	0	1	0	0	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	0	10	3	1	1	0

Once again the SQ makes sure the SX has enough room in the Parameter cache before it can pick up this thread.

```
Execute_end 0 Alu 0 Tex 1 Alu 0 Alu
```

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
5	1	0	0	0	1	0	100	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	TEX	1	0	0	0	1	1	0

This executes on the TP and then returns:

State Bits

CFP	ECM	LI	CRP	PB	EXID	GPR	EB	CPTR	LOD
5	2	0	0	0	1	0	100	0	0

Status Bits

VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
1	ALU	1	1	0	0	1	1	1

Waits for the TP to return because of the textures reads are pending (and SERIAL in this case). Then executes and does not return to the RS because the LAST bit is set. This is the end of this thread and before dropping it on the floor, the SQ notifies the SX of export completion.

24. Open issues

Need to do some testing on the size of the register file as well as on the register file allocation method (dynamic VS static).

Saving power?

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	13 November, 2000	4 September, 2015	GEN-CXXXXX-REVA	1 of 16

Author: Steve Morein

Issue To:

Copy No:

R400 Architecture Proposal

ver 0.1

Overview: The is a proposal for the overall architecture of the R400. It is also just a proposal, and nothing is decided yet.

AUTOMATICALLY UPDATED FIELDS:

Document Location:	VST FireWire HD:r400 spec
Current Intranet Search Title :	R400 Top Level Spec

APPROVALS	
Name/Dept	Signature/Date

Remarks:

THIS DOCUMENT CONTAINS CONFIDENTIAL INFORMATION THAT COULD BE SUBSTANTIALLY DETRIMENTAL TO THE INTEREST OF ATI TECHNOLOGIES INC. THROUGH UNAUTHORIZED USE OR DISCLOSURE.

Table Of Contents

1. FEATURES 6
1.1 AGP8x and possibly serial AGP 6
1.2 128 Bit memory interface 6
1.3 Nearly transparent dual chip 6
1.4 Unified processing pipe 6
1.5 Front end scaling 6
1.6 Control processor 7
1.7 Real-Time drawing command ability 7
1.8 3D Features 7
1.8.1 Noise Textures 7
1.8.2 Shadow buffers 7
1.8.3 Anti-Aliasing 7
1.8.4 Texture compression 7
1.8.5 Z compression 7
1.8.6 Filtering 8
1.8.7 Curved Surface Support 8
1.9 High color depth 8
1.10 Performance 8
2. AREA 8
3. SCHEDULE 8
4. PROCESS 8
5. DUAL CHIP 9
6. GENERAL RENDERING OPERATION 9
6.1 Unified Shader 9
6.2 3D Rendering 9
6.3 2D Rendering 11
6.4 Real-time Rendering 12
7. DISPLAY OPERATION 12
8. BLOCK DIAGRAM 13
9. SHORT BLOCK DESCRIPTIONS 14
9.1 SYS 14
9.1.1 HBIU 14
9.1.2 HDP 15
9.1.3 MISC 15
9.1.4 Rom 15
9.1.5 VIP 15
9.1.6 I2C 15
9.1.7 ? 15
9.1.8 ClockGen 15
9.1.9 CP 15
9.1.10 RBBM 15
9.1.11 MC 15
9.2 Display 15
9.3 Grfx 15
9.3.1 PrimitiveAssembly/vertex cache 15
9.3.2 Raster Engine 15

9.3.3 Sequencer 15
9.3.4 Datapath 15
9.3.5 TextureEngine 15
9.3.6 RenderBackend 15
10. TOP LEVEL INTERCONNECTIONS 15
10.1 First Level Sub Heading 15
10.1.1 Second Level Sub Heading' 15

Revision Changes:

Rev 0.0 (Steve Morein) Date: November 6, 2000 Initial revision.	Document started
Rev 0.01 Steve Morein Date: November 10,2000	Document continued

Introduction

This document outlines a proposal for the r400 architecture.

A minor note: in the middle of writing this I decided that it makes the most sense to call the “pixel” pipelines shader pipeline since they handle vertices and pixels. I have not gone through this to make sure that my usage is consistent.

1. Features

1.1 AGP8x and possibly serial AGP

The R400 will at a minimum support AGP4x and AGP8x interfaces. We may also support 3.3V i/o including AGP2x and 3.3V PCI. We need to consider how we interface to LDT (AMD) and possibly the Motorola rapid I/O that may be used in future Apple Designs (G5).

1.2 128 Bit memory interface

We are thinking of only supporting a 128 bit interface to memory. The memory will be configured as four channels of 32 bits each. The atomic fetch until will be 256 bits in expectation that some high speed memories will use a prefetch-8 architecture. Logic in the memory controller will optimize down to 128 bit writes when possible on DDR or prefetch 4 memories. Memories up to 500 MHz will be supported (1 gigabit data rate).

Memory is the most open issue on the R400. We need to develop a roadmap ASAP for how memory will develop, and this may significantly affect our plans.

1.3 Nearly transparent dual chip

To be able to address the very high end desktop/enthusiast market we will support a glueless two chip design instead of a 256 bit bus. Unlike previous dual chip designs we have done, this is targeted to be a mainstream product. This implies that it can easily be WHQL'd, and can accelerate all applications and benchmarks, not just a subset of full screen apps. A separate document outlines the two proposals we are looking at for the dual chip design.

There will be costs added to the base chip to support this. Design time, pins, and area will be impacted by adding this support.

1.4 Unified processing pipe

The most ambitious feature in this design is the "truly unified pipe" : a single programmable pipeline is used for 2D, Video, 3D vertex, and 3D pixel operations. The unified pipeline does all of its calculations in 32 bit floating point, the same as the existing vertex transform in previous chip, and the next step in the precision of the color/pixel calculations which have increased from 8 bits (R100), through 16 bits (R200), to the 20 bits in the R300.

There is an area cost to the unified pipeline since we are forced to go to 32 bit precision for color, when application requirements may need less (22 to 24 bits). However the unified pipeline results in a single math/register structure compared to the separate structures in a more traditional design. It is hoped that by only needing to design the one structure we can make the investment in design time and effort to really optimize the area.

Some of the benefits to merging the pipelines include allowing the vertex operations to do texture fetches, which we could not afford add logic to the transform pipe to do, a single programming model for both operations, more precision on color than we would normally provide, and the ability to support significantly more registers and instructions in pixel shaders.

One important benefit is load balancing. In the current pipeline when the app it transform bound the pixel pipeline is idle some significant portion of the time, and when the app is raster bound the transform hardware idle. The unified pipeline presented here dynamically allocates its processing power between transform and raster.

1.5 Front end scaling

We will remove the back end scaling capability from the display, and replace it with a non-scaling overlay. This will require us to be able to implement scaling using the unified pipeline. Key features that will need to be supported are large filter kernels, de-interlacing, frame rate conversion, and good support for YUV and color conversion.

1.6 Control processor

To allow us to emulate a backend scaler and to enable new applications the control processor will be enhanced with event based streams. These are secondary, real time, command streams that start execution when an event happens.

1.7 Real-Time drawing command ability

To allow for the emulation of backend scaling as well as support new features we need to be able to interrupt the 3D pipe and be able to execute high priority commands with low latency. At the moment it appears far to difficult to be able to insert a new command at the top of the 3D pipeline and meet latency requirements (which I believe we wish to define as around a 1/16 of a frame refresh). This would require us to be able to interrupt triangles in the midst of rasterizing, inset vertices in the midst of a large vertex array, and other nasty things. I think instead we can get by with a second rasterizer which drives the pixel pipelines. Setup would be done with software, but since the majority of the real time rasterization is expected to be simple

1.8 3D Features

There are a number of new 3D features we are considering for inclusion. Additional features may be added, and some of these may be dropped.

1.8.1 *Noise Textures*

Perlin style noise is useful for a number of applications. It is generated on chip and consumes no external memory bandwidth. It also larger than any physical texture can be: 256x256x256 lattice points, and still has detail when the resolution is 4Kx4Kx4K. There is an opportunity to get this adopted as part of dx9.

1.8.2 *Shadow buffers*

John Carmack is using shadow volumes to generate shadow effects in doom3. Shadow volumes are very poor way to use modern 3D pipelines. (will add more detail here later). Shadow buffers have two key limitations: very high resolutions are required to avoid aliasing, and traditional shadow buffers can not be mip-mapped so filtering is real problem. Through a combination of the z techniques we have developed and, hopefully, deep shadow buffers, we can solve both of these problems and widely enable shadow buffers.

1.8.3 *Anti-Aliasing*

We want to further improve the anti-aliasing used in the R300 by reducing the needed memory, and possibly increasing the number of samples per pixel. The goal is more than fifty percent of the performance and less than three times the memory of anti-aliased rendering. We should also look into improved methods.

1.8.4 *Texture compression*

To further reduce bandwidth we need to improve texture compression. We need to achieve both better compression that S3TC, and have a high enough quality that textures that would lose too much detail with S3TC can be compressed. Both of these goals do not need to be achieved simultaneously on all textures. We also need to look at compression of non-traditional surfaces such as normal maps.

1.8.5 *Z compression*

We will build on the R300 slope based compression but we are looking at supporting maxmin for cachelines that do not compress with slopes (either too many slopes per cacheline, or the pixel shader modifies the z value)

1.8.6 *Filtering*

As part of the move to front end scaling we need better than bi-linear filters. Goals are : arbitrary sized separable filters and 4x4 bi-cubic. Being able to support programmable weights is nice.

1.8.7 *Curved Surface Support*

We need to figure out how we are going to support curved surfaces in this architecture. I think that we can find a way to use the wide ability of a vertex shader to implement acceleration for subdivision surfaces, but the vertex only level of processing in the shader pipeline means that something ahead of it needs to set up the surface. At one point I imagined that we could use the sibyte processor as a CP, which would have the power to do the curved surface setup. That is obviously no longer possible.

1.9 High color depth

We will support a 64 bit color buffer (16:16:16:16), the exact format (fixed, floating, etc.?) has yet to be decided

1.10 Performance

I think we can increase the clock speed from 300 MHz to 500MHz.
Historically the goal has been to double speed in each generation, assuming a constant clock speed. However since we are considering the dual chip solution for the very high end we may not need to be 2x the speed of the R300. Our use of a 128 bit memory bus instead of 256 bits will impose a potentially lower bandwidth.

That said I would still like to aim for 2x the internal processing capability of the R300:

	R300	R400
Clock speed	300 MHz	500 MHz
Pixels per clock	8	16
Bi tex fetches per pixel	2	2
ALU ops/clk (mac’s)	64 (dedicated)	192 (shared)
Peak Tri/Sec	150	250
Peak xform fp ops/clk	16? (dedicated)	192 (shared)

We may need to reduce this performance goal to meet our area goal.

2. Area

The area goal for the R400 is 10mm on a side in .13 micron CMOS
There will probably be a lower cost version with a target area of 8.5 on a side.

3. Schedule

Tapeout	April 2, 2002
Samples	May, 2002
Production	Nov, 2002

4. Process

At the moment this looks like an easy choice: .13 will be in production for over a year, and .10 does not show up until the very end of 2002 according to the TSMC and UMC roadmaps.

We will probably want to be in a flip chip packaging approach to meet power distribution goals. It will also reduce the cost of the dual chip option by making the extra pins needed for the interface cheaper. Is there a way to have an option to wire bond it also? Possibly without the dual chip interface, and with less pwr/gnd forcing a lower clock speed. This may make sense for a lower cost sku.

5. Dual Chip

<need to copy the dual chip notes over and add to them>

6. General rendering operation

6.1 Unified Shader

The unified shader is a simd/vector engine that performs the same instructions on four sets of four (16 total) elements. For pixel shader operations the elements are pixels with the sets of four required to be 2x2 footprints. For vertex shader operations the sixteen elements are sixteen vertices. The basic element is a 4 value vector – frequently interpreted as x,y,z,w or r,g,b,a.

The user model for the unified shader is composed of a variable number of general purpose registers, a subset of which are usually initialized with data. An ALU can do simple math, conditional moves, and permutations on the registers, and the ability to do a limited number of memory reads using the texture cache. The number of register is variable, and the number of registers required for an operation are specified when the task is submitted to the unified shader. The unified shader will not start the task until there is enough free room for the tasks registers.

The unified shader is based on the R300 partially unified shader.

6.2 3D Rendering

For 3D rendering data is passed twice through the unified shader- once to transform the vertices and a second time to determine the color of the pixels.

The input to the 3D pipe is expected to be indexed vertex arrays. Linear vertex arrays can easily be supported by the CP generating sequential indices. Inline vertex data is an open issue, I would prefer to write it to memory and then fetch it as a vertex array rather than add a direct path.

The stream of indices is sent to the Primitive Assembly block by the CP. The front of the primitive assembly block maintains the tag for the vertex cache; The vertex cache stores transformed vertices. As misses are detected in the tag, the indices that miss are placed into 16 entry vectors. Each vector contains a state pointer, a pointer to the vertex shader to be used, and the 16 indices to vertices that need to be transformed. When either a vector is filled with 16 entries or a state change happens (so that the next vertex does not share the state and vertex shader with the previous vertex) the vector is issued to one of the “shader” pipelines for transformation. Which of the four shader pipelines it is issued to determined either by some effort of load balancing or a simple round robin. All that is submitted to the pixel pipeline is the state, the vertex program, and the indices. The shader pipeline will fetch the vertex array data through the cache infrastructure that is also used for texture fetches. After the tag the indices (actually now the indices into the vertex cache) are placed into a latency FIFO to hide the latency of transforming the vertices.

The shader pipeline receives the vector of 16 indices from the primitive assembly block. The shader pipeline operates, when rendering pixels, by processing a vector of four 2x2 pixel footprints, A total of 16 pixels. For vertex processing each of the pixels is replaced with a vertex. The vertex program includes information of how many local variables it will need. The rasterizer waits until that many local variables are free, (as each executing thread in the shader pipeline terminates it frees its local variables). With the proposed shader datapath the maximum number of local variables per vertex is 256. However this leaves no ability to hide latency, 16 to 32 local variables will probably maximize latency hiding and therefore performance. The vertex shader program can use all the capabilities of the shader pipeline including texture fetches and dependent lookups. At the end of the vertex program, the transformed coordinates must be output. One output will be the x,y,z,w position which we be stored in the position cache of the vertex cache. The vertex program may also output a number of parameter values (colors, texture coordinates, other interpolated inputs into the pixel shader). The parameter values must be output as a multiple of four 128 bit words, as the parameter cache is designed for this.

The primitive assembly block reads the indices back out of the latency FIFO and accesses the position cache portion of the vertex cache. It assembles the vertices into primitives (lines, triangles, rectangles, quads?, points, ?). Baricentric values are assigned to the vertices, and will be used later in the rasterizer to interpolate the parameters. The parameters are not accessed by the primitive assembly logic, which only works from the position data. The primitive is clipped against both the viewing volume as well as user clip planes, with fractional baricentric coordinates assigned to the clipped primitive sections. The primitive goes through the perspective divide and the viewport transform. The resulting screen space primitive is setup (plane equations for 1/W, Z, and the baricentric coordinates). The resulting primitive data, including the indices back into the parameter portion of the vertex cache are broadcast to the four pipes. The final time that an index is output that access the oldest vertex cache line, a token is also sent. When all of the four pipelines return the token the primitive assembly block can free that cacheline and allow it to be used for a new vector of vertices. The performance goal in the primitive assembly block is a triangle every two clocks.

Each pipe has a FIFO in front of the rasterize to load balance. Each pipe will handles 16x16 sections which are interleaved between the pipes. To maximize the effective size of the FIFO we will probably cull the triangle list before the FIFO. The rasterizer will request the parameter data from the parameter cache for the primitives. A small latency hiding FIFO will hide the latency of the access to the parameter cache. The parameter cache is 512 bits wide, and the interfaces from the parameter cache to the rasterizer are 128 bits wide, this allows the parameter cache to output one pipelines request per clock, which is serialized over four clocks, keeping all four interfaces busy. The rasterizer keeps a small cache of three to four vertices, this allow only the new parameter to be fetched when adjacent triangles are processed. The parameter cache interface imposes a second performance limits, in the worst case each polygon covers all four pipelines and there are no vertices shared from triangle to triangle. In this case the peak performance is (500 MHz / (4 pipelines * 3 vertices) = (500/12) = 41.6 million triangles per second. In the best case triangles are perfectly stripped and never cross over pipeline boundaries. In this case the peak performance (If we ignore the setup limit) is 500 million triangles per second. As a practical manner we should be able to approach the setup limit of 250 Million triangles per second.

The rasterizer also contains a portion of the hierarchical Z memory. We are looking into moving this into a cache based approach, but that is far for certain at this point. We would like to be able to do heirz culling at a speed in excess of 64 pixel per clock per pipelines (256 pixels per clock total). We are also going to consider some of the improved latency heriz options to improve culling efficiency.

The rasterizer will generate four pixels per clock if there are no more than eight interpolated parameters. The rasterizer generates vectors of four 2x2 footprints (16 pixels). Each 2x2 footprint must be screen aligned and from the same triangle (with a single shared z slope). The four footprints only need to share the same state and shader program.

Before starting the processing of a vector the rasterizer (which includes the sequencer for the shader pipeline) checks to make sure that there are enough free registers in the shader pipeline for the pixel shader program. If not, it stalls until there are enough. The rasterizer also needs to arbitrate between the three streams of vectors to be shaded: the vertex stream, the pixel stream, and the real time stream. I think it will be sufficient for the real time stream to have priority over the vertex stream which has priority over the pixel stream. This will meet the realtime demands, and keep the vertex cache filled.

The vector is then processed by the shader pipeline. We will probably support up to eight sequentially dependent texture fetches. (to use the R300 terminology, eight clauses). 16 (8?) textures are supported, but each texture can be accessed multiple times by a single pixel shader which can provide a different address each time. This is especially useful for complex filters.

The output of the pixel shader is the final color of the fragment. The pixel shader may also replace the Z value. Fog and stippling must be done in the pixel shader program.

The render backend does the z compare, stencil operation and color alpha blend.

The texture fetch path has a number of design options. One option is an approach where the local, multiported, texture cache is small (1 to 4 KB), and contains uncompressed color in a canonical format (32 bits per pixel) and uses a 4x2 or 4x4 cacheline. This is backed up by a large (>16KB) L2 cache which also stored uncompressed 8x8 cachelines. The decompression logic lives between the memory controller and the L2 cache.

An alternative design uses the L2 cache to contain data in memory format (compressed) which is decompressed as needed to fulfill L1 texture cache misses. This will increase the effective size of the L2. The L2 cache is distributed, with 1/4 of it residing in each memory controller. The Texture decompression logic can either be located in each shader pipeline, or exist as a shared block(s) that receive data from all four memory controller and send the decompressed 4x4 cachelines to each shader pipeline. The unified decompression block will result in better performance, and possibly less area, at the cost of some of the scalability.

Assuming that we chose the L2 in memory controller and the unified decompression logic, the texture path would work as follows:

In a four pipeline design there are two texture decompression blocks, one for the “left” texture units in each shader pipeline, and the second for the “right” texture units. In the two pipeline, lower cost, version of the chip only a single decompression pipeline is used, serving the left and right texture units.

The L1 texture cache receives a texture request from its shader pipeline. The usual tag and latency FIFO is used to generate the misses. These are sent to the shared texture decompression block, which looks up the texture to find the physical address and then sends the request to the L2 cache in the memory controller. The L2 also has a latency FIFO and tag, and will return the data in order (but there is no order guaranteed between the data returning from each L2). The decompression block has a buffer which is used to place the data from the memory controllers back in order. The decompression logic decompresses the texture and returns, in order, the 4x4 cachelines that the L1 caches are requesting. Most of the compression techniques we are considering are based on an 8x8 tile (or 4x4x4), when necessary the decompression logic will decompress an entire 64 pixel tile and only return the requested 16 pixels to the L1 cache. This will tend to increase the bandwidth between the decompression logic and the L2 cache as 8x8 blocks are repeatedly requested to provide different 4x4 subtiles to the L1. The L2 cache will prevent the repeated reads from going to memory, and we will probably implement an “L0” style cache in front of the L2 to also catch the redundant requests.

Each memory controller will have two 64 bit read return buses, one to each of the two decompression blocks, each decompression blocks drives a separate 128 bit bus to each of the four shader pipelines. This will tend to have better utilization and load balancing than having the memory controller drive a 32 bit bus to the decompression logic in each shader pipeline. While the total number of wires is similar (128 bits per memory controller, 128 bits into each texture cache) we are less likely to leave the texture pipes starved when there is some imbalance.

6.3 2D Rendering

2D rendering is implemented in the 3D pipeline. The first reason for the change is performance; The current 2D pipeline can render 128 bits per clock (16 8 bit pixels, 8 16 bit pixels, 4 32 bit pixels). The 3D pipe can render 16 pixels per clock, and the pixels can be 8 to 64 bits wide. Secondly routing the three busses needed by the 2D engine (src read, dest read, dest write) has a certain cost, and complicated the design of the chip. If we attempt to improve the performance of the 2D pipe these busses will increase in size, further complicating the design. Another reason is dual chip rendering, it would be nice if 2D as well as 3D operations increase in speed. (2D operations include things like color clears and texture uploads that do show up in 3D benchmarks.

There are four issues currently known that will affect placing the 2D commands into the 3D pipelines:
1) Command compatibility
2) Hostpath blits
3) ROP3
4) Overlapping blits.

We wish to be as compatible as possible with the existing PM4 2D model. This will require that the CP be enhanced, allowing it to translate 2D commands into commands understood by the 3D pipe. We will ad interfaces to the 3D pipe to make this easier but there will still be significant amount of work that still needs to be done by the CP.

Host path blits can no longer work as they do now. Four pipelines will be attempting to execute the requested command in parallel, walking the area to be drawn in some tightly tiled pattern to optimize memory and cache performance. This bears little resemblance to the single linear stream of data from embedded into the PM4 command stream. In addition the shader pipelines are heavily optimized for a pull, “reverse” mapping model, and not a push, “forward” model. The basic solution to this problem is for the 3D pipes to pull the source data directly out of a command/data buffer, in whatever order, and in whatever parallel streams exist. Whether we give the PM4 engine the ability to skip over the hostpath data, or force the driver to move hostpath data into a second command ring still needs to be decided. It is possible that one or more changes may be needed to the driver for this.

2D supports a ROP3 operation that requires the destination color as well as two sources: the source, and the pattern. To support this the pattern color is output by the pixel shader on the Z output which is not used by 2D operations. The render backend now has all three needed sources.

Overlapping blits is an ugly problem that I have not yet found an acceptable solution to. More work needed.

One benefit to these changes is the 2D operations will also be accelerated by the second chip in a dual chip board.

6.4 Real-time Rendering

7. Display operation

The display must be able to display from microtiled surfaces and overlays. This will generally force us to adopt line buffers.

The display should support at least two outputs, ideally we will be able to support two high resolution outputs and a low resolution output (TV out)

We will drop support for overlay scaling, and therefore supporting an overlay on all displays becomes affordable, fixing a “bug” that our current dual display products suffer from.

We will place the overlay line buffers in the memory controllers, this changes the interface from the memory controllers to the display from a wide “bursty” interface to a narrow continuos interface.

<this rest of this was copied out of another document and needs some editing to fit in this document. Bear with us as construction of this document continues>

Video buffer operation:

At beginning of even scanline (scanline 0) 1/2 of line buffer is filled.
The upper half of the buffer is read out, at the same time as the second half of the buffer is filled. When the scanout reaches the midpoint 3/4 of the buffer is filled. When the scanout reaches the end of the scan the buffer is filled. The speed at which the buffer is filled must be greater than 1/2 of the rate at which the buffer is scanned out.

For the odd scanlines, the buffer is completely filled at the start of scanout (as a result of the even scanline finishing properly). As the lower half is scanned out, reads are issued to fetch the data for the next pairs of scanlines. At the end of the odd scanline, the buffer is expected to contain half of the data for the next scanline.

Another way of looking at this is as follows:

At the beginning of the odd scanline the scan buffer is filled. As each word is read from the buffer and sent to the display logic, a request is made to the memory controller to fill in the data. It is not necessary for all the data for the next scanline pair to be fetched by the time that the scanline reaches the end, the real requirement is for the last word in the scanline buffer to get there just before it is read, at the end of the next even scanline.

We will support a single non-scaled overlay per each display.

Some bandwidth numbers
In reality we do not need to deal with quite as much bandwidth as the FIFO in the display can hide the horizontal retrace.
350 MHz primary fetch, 32 bit data:
350 MHz primary display, 32 bits
350 MHz overlay fetch

350 MHz overlay display
total: 350 MHz * 16 bytes (128 bits)

Since we support dual monitor, this is doubled.

One design option is to split the display scanline into pieces and move them into the memory controller. This greatly reduces the exposed bandwidth in the system (reducing power and routing problems)

If we assume that there are four memory controllers, each with 2GB/s of memory bandwidth then the following will work:

core clock speed: >= 1x the memory clock
memclk = 500 MHz
coreclk = 500 MHz

each memory interface is 32 bits at a DDR rate, and the fetch granularity is 256 bits.
Therefore if data was continuously received into the display FIFO 64 bits would be received every clock. A 256 bit interface at the core clock rate is more than adequate..

the memory size needed for two 2048 displays is : 2048*4 bytes * 2 scanlines * 4 buffers is 64KB. So each buffer is 16KB (128 Kbit). With a 256 bit interface the memory is 256x512 single ported.

For writes into the write buffer as a result of memory fetches, a small buffer reorders data between pairs of 256 bit words so that while what is read from memory is 256 bits containing two vertically stacked 128 bit words, what is written is two 128 bit words that are on the same scanline.

The interface from the memory controller to the display only needs to be big enough for the sustained bandwidth, and not the peak memory speed bandwidth. A 16 bit interface to each display seems like more than enough.
(compare this to the rage 6 with two 128 bit busses between the memory controller and display)

A couple more notes:
for the most part the memory format is stored in the scanline buffer. The exception is 64 bit, which we would like to convert to something like 11:11:10 or 8:8:8:8 This may mean some sort of gamma circuit in the memory controller.

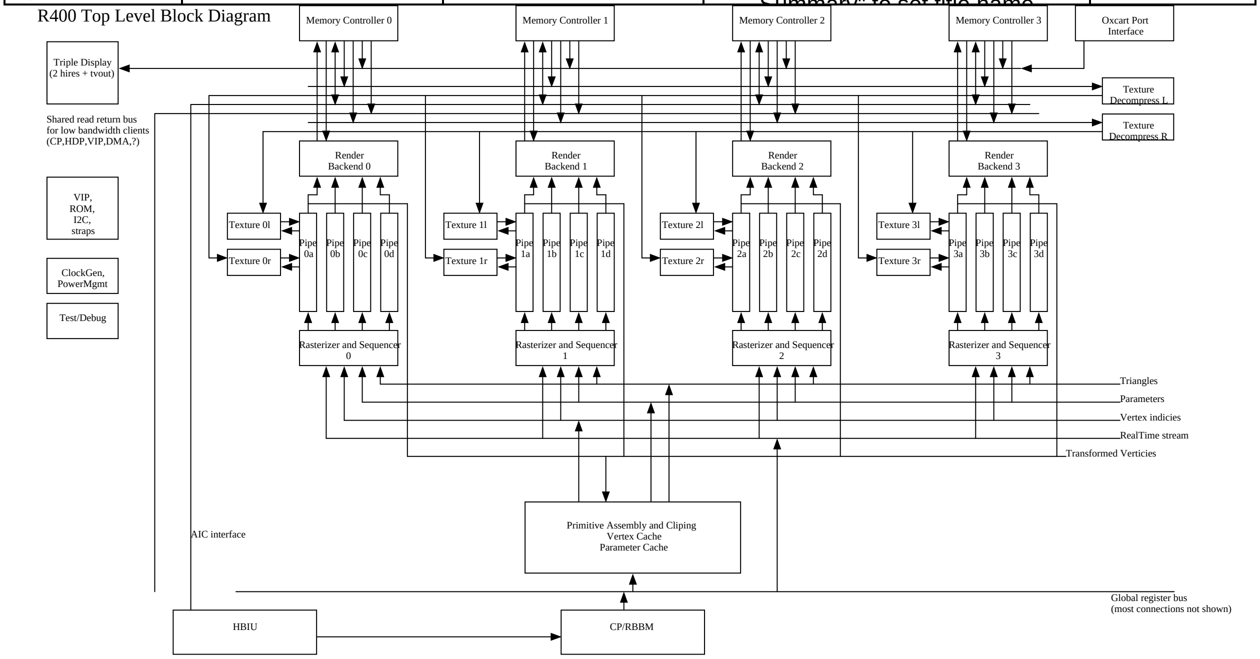
A LUT would exist in the display for gamma de-correction and pallet support.

The interleave between the memory controllers would have to be compatible with the tiling and still give good performance.

A big question is how does this work in a two chip board? I had been thinking about interleaving on a fine basis between the chips with a display controller in one chip fetching from both, but this somewhat flips that around. We need to route the video signals as an extra channel between the chips, this will add complexity, but it actually is less bandwidth since the overlay is combined first.

8. Block diagram

R400 Top Level Block Diagram



9. Short Block descriptions

9.1 SYS

The system blocks support the chip, but are not graphics specific.

9.1.1 *HBIU*

The HBIU is the interface to the host bus. It implements four interfaces: register/read write, HDP for host access to memory and the

9.1.2 *HDP*

9.1.3 *MISC*

9.1.4 *Rom*

9.1.5 *VIP*

9.1.6 *I2C*

9.1.7 *?*

9.1.8 *ClockGen*

9.1.9 *CP*

9.1.10 *RBBM*

9.1.11 *MC*

The memory controller is distributed, each of the four memory channels has a separate memory conttoller. Each memory control contain a part of the L2/Line buffer memory. This large buffer serves a number of purposes in the graphics chip, including L2 cache for textures and verticies.

9.2 Display

9.3 Grfx

9.3.1 *PrimitiveAssembly/vertex cache*

9.3.2 *Raster Engine*

9.3.3 *Sequencer*

9.3.4 *Datapath*

9.3.5 *TextureEngine*

9.3.6 *RenderBackend*

10. Top Level Interconnections

10.1 First Level Sub Heading

10.1.1 *Second Level Sub Heading'*

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	11 March, 2001	4 September, 2015	GEN-CXXXXX-REVA	1 of 32

Author: Steve Morein

Issue To: **Copy No:**

R400 Top Level Specification

ver 0.2

Overview: This replaces the R400 architecture specification.

AUTOMATICALLY UPDATED FIELDS:

Document Location:	Document1
Current Intranet Search Title :	R400 Top Level Spec

APPROVALS

Name/Dept	Signature/Date

Remarks:

THIS DOCUMENT CONTAINS CONFIDENTIAL INFORMATION THAT COULD BE SUBSTANTIALLY DETRIMENTAL TO THE INTEREST OF ATI TECHNOLOGIES INC. THROUGH UNAUTHORIZED USE OR DISCLOSURE.

Table Of Contents

1. FEATURES 7
1.1 AGP 8x 7
1.2 256 Bit Memory Interface 7
1.3 Unified Processing pipe 7
1.4 Front end scaling 7
1.5 Real-Time drawing command ability 7
1.6 3D Features 8
1.6.1 Noise Textures 8
1.6.2 Shadow buffers 8
1.6.3 Sort Independent Transparency 8
1.6.4 Anti-Aliasing 8
1.6.5 Texture compression 8
1.6.6 Z compression 8
1.6.7 Texture Filtering 8
1.6.8 Curved Surface Support 9
1.6.9 Displacement maps 9
1.7 High color depth 9
2. PERFORMANCE 9
3. SCHEDULE 9
4. PROCESS 9
5. GENERAL CHIP OPERATION 9
5.1 Unified Shader 9
5.2 3D Rendering 10
5.3 Real Time Rendering 12
5.4 State Management 13
5.5 Bad Data 13
5.6 Display operation 14
6. BLOCK DIAGRAM 15
7. BLOCKS 15
8. BLOCK DESCRIPTONS 16
8.1 HBIU – host bus interface unit 16
8.1.1 Description 16
8.1.2 Major interfaces 16
8.1.3 Block diagram 17
8.2 CP – control processor 18
8.2.1 Description 18
8.2.2 Major interfaces 18
8.2.3 Block diagram 19
8.3 RBBM – register interface manager 19
8.3.1 Description 19

8.3.2 Major interfaces 19
8.3.3 Block diagram 20
8.3.4 RBBM operation 20
8.4 CLK – clock generator 21
8.4.1 Description 21
8.4.2 Major interfaces 21
8.4.3 Block diagram 22
8.5 TC – test controller 22
8.5.1 Description 22
8.5.2 Major interfaces 22
8.5.3 Block diagram 22
8.6 VIP – Video input port 22
8.6.1 Description 22
8.6.2 Major interfaces 22
8.6.3 Block diagram 22
8.7 ROM – boot rom 22
8.7.1 Description 22
8.7.2 Major interfaces 22
8.7.3 Block diagram 22
8.8 I2C – I2C interface 22
8.8.1 Description 22
8.8.2 Major interfaces 22
8.8.3 Block diagram 23
8.9 DU – Display 23
8.9.1 Description 23
8.9.2 Major interfaces 23
8.9.3 Block diagram 24
8.10 MH – Memory Hub 24
8.10.1 Description 24
8.10.2 Major interfaces 24
8.10.3 Block diagram 24
8.11 HDP – Host Data Path 24
8.11.1 Description 25
8.11.2 Major interfaces 25
8.11.3 Block diagram 25
8.12 IDCT – Mpeg decoder 25
8.12.1 Description 25
8.12.2 Major interfaces 25
8.12.3 Block diagram 25
8.13 PA – Primitive Assembly 25

8.13.1 Description 25
8.13.2 Major interfaces 25
8.13.3 Block diagram 26
8.14 TD – Texture Decompression 26
8.14.1 Description 26
8.14.2 Major interfaces 27
8.14.3 Block diagram 27
8.15 RE – Raster Engine 27
8.15.1 Description 27
8.15.2 Major interfaces 28
8.15.3 Block diagram 29
8.16 SP – Shader Pipe 30
8.16.1 Description 30
8.16.2 Major interfaces 30
8.16.3 Block diagram 31
8.17 TP – Texture Pipe 32
8.17.1 Description 32
8.17.2 Major interfaces 32
8.17.3 Block diagram 32
8.18 RB – Render Backend 32
8.18.1 Description 32
8.18.2 Major interfaces 32
8.18.3 Block diagram 32
8.19 MC – Memory Controller 32
8.19.1 Description 32
8.19.2 Major interfaces 32
8.19.3 Block diagram 32
9. COMMON FOUNDATIONS 32
9.1 Logic Design 32
9.1.1 Data formats 32
9.1.2 Register Bus 32
9.1.3 Block Communication protocol 32
9.2 Software 32

Revision Changes:

Rev 0.0 (Steve Morein) Date: March 11, 2001 Initial revision.	Document recreated from earlier documents
Date March 14,2001	Finally got back to editing it.

Introduction

The R400 will be the high end standalone graphics chip product when it is introduced.
It will be followed very rapidly with two variants:
The RV400, aimed at the volume PC space
The R450, aimed at a volume high end market.
The targets for the three chips are:

Part	Clock Speed	pixels/clk	texture fetches/clk	alu ops/clk	Memory width	Memory speed	die size	Tapeout
R400	400 MHz	8	16	32	256	400MHz	11.5	July,2002
RV400	500 MHz	4	8	16	128	500 MHz	8.5	Nov 2002
R450	500 MHz	8	16	32	256?	500 MHz	9.5	Feb 2003

1. Features

1.1 AGP 8x

The chip will support the 32 bit AGP interface at speeds up to 8x. I expect that we will need to support AGP 1x and 2x which require 3.3 Volt I/0 (AGP 4x is 1.5v and AGP 8x is 750mv). AGP fast writes are supported for access to the frame buffer.

Open issue: 64 bit address space support.

1.2 256 Bit Memory Interface

The R400 and R450 support four memory channels, which can be 32 or 64 bits wide; the maximum memory bus width is a total of 256 bits. The RV400 supports two memory channels and a maximum total width of 128 bits.

All channels need to be configured identically, 1, 2 or 4 channels can be configured.

Memory standards supported:

I/O	Voltage	Memory type	Speed
SSTL2.5	2.5	DDR	100 to 500 MHz
SSTL1.8	1.8	DDR/infineon	300 to 500 MHz
Elpida	1.8 (1.5?)	Elpida	300 to 400 MHz
Infineon	1.2, 1.0 V	Infinion e-dram	500 MHz

No support for SSTL3.3, or SDRAM (LVTTL – 3.3V) is planned.

1.3 Unified Processing pipe

The most ambitious feature in this design is the “truly unified pipe” : a single programmable pipeline is used for 2D, Video, 3D vertex, and 3D pixel operations. The unified pipeline does all of its calculations in 32 bit floating point, the same as the existing vertex transform in previous chip, and the next step in the precision of the color/pixel calculations which have increased from 8 bits (R100), through 16 bits (R200), to the 20 bits in the R300.

There is an area cost to the unified pipeline since we are forced to go to 32 bit precision for color, when application requirements may need less (22 to 24 bits). However the unified pipeline results in a single math/register structure compared to the separate structures in a more traditional design. It is hoped that by only needing to design the one structure we can make the investment in design time and effort to really optimize the area.

Some of the benefits to merging the pipelines include allowing the vertex operations to do texture fetches, which we could not afford add logic to the transform pipe to do, a single programming model for both operations, more precision on color than we would normally provide, and the ability to support significantly more registers and instructions in pixel shaders.

One important benefit is load balancing. In the current pipeline when the app it transform bound the pixel pipeline is idle some significant portion of the time, and when the app is raster bound the transform hardware idle. The unified pipeline presented here dynamically allocates its processing power between transform and raster.

1.4 Front end scaling

We will remove the back end scaling capability from the display, and replace it with a non-scaling overlay. This will require us to be able to implement scaling using the unified pipeline. Key features that will need to be supported are large filter kernels, de-interlacing, frame rate conversion, and good support for YUV and color conversion.

1.5 Real-Time drawing command ability

To allow for the emulation of backend scaling as well as support new features we need to be able to interrupt the 3D pipe and be able to execute high priority commands with low latency. The point of interruption is in the primitive assembly, the maximum latency will be about the time it takes to render 4096 pixels. The real time commands are inserted into the 3D pipeline after transform, clipping, and setup. Those function need to be performed by the driver. There are also limits on the number of constant registers available.

1.6 3D Features

There are a number of new 3D features we are considering for inclusion. Additional features may be added, and some of these may be dropped.

1.6.1 *Noise Textures*

Perlin style noise is useful for a number of applications. It is generated on chip and consumes no external memory bandwidth. It also larger than any physical texture can be: 256x256x256 lattice points, and still has detail when the resolution is 4Kx4Kx4K. There is an opportunity to get this adopted as part of dx9.

1.6.2 *Shadow buffers*

John Carmack is using shadow volumes to generate shadow effects in doom3. Shadow volumes are very poor way to use modern 3D pipelines. (will add more detail here later). Shadow buffers have two key limitations: very high resolutions are required to avoid aliasing, and traditional shadow buffers can not be mip-mapped so filtering is real problem. We are able to solve the first problem through a combination of our improved anti-aliasing Z compression, and a new method of implementing the shadow map probe.

1.6.3 *Sort Independent Transparency*

We are currently looking into how best to support sort independent transparency. The two plans are either the dual Z buffer approach, or the approach described in <need to decide where the email should be placed so others can see>

1.6.4 *Anti-Aliasing*

The changes from the R300 include an increased number of samples per pixel, probably eight, and support for an allocated frame buffer size smaller than the worst case maximum.

1.6.5 *Texture compression*

To further reduce bandwidth we need to improve texture compression. We need to achieve both better compression that S3TC, and have a high enough quality that textures that would lose too much detail with S3TC can be compressed. Both of these goals do not need to be achieved simultaneously on all textures. We also need to look at compression of non-traditional surfaces such as normal maps. Advances here are dependent on the availability of resources to work on this. If we are unable to find resources we will support the s3tc compression currently in D3D.

1.6.6 *Z compression*

<larry needs to give me a paragraph here>

1.6.7 *Texture Filtering*

The texture pipes can fetch a 2x2 region from the texture map and filter it.
The data per pixel can either be four eight bit values, two sixteen bit values, or one 32 value. All data needs to be fixed point.
Linear filters are completely built in, and it takes 1 cycle for bi-linear, 2 for tri-linear, four for quadra-linear (filtered mip-mapping of volume textures). Variable depth anisotropy is supported in hardware with the texture pipe calculating the number of samples needed. Optionally the pixel shader can calculate the number of samples, and how to increment the texture address, and provide this to the texture pipe.

1.6.8 *Curved Surface Support*

We will support curved surfaces through combination of vertex shader code and a tessellation engine to generate new vertices.

The tessellation engine generated new vertex indices from a input vertex index array. The new indices contain both the coordinate in parametric space of the vertex, and the indices to the surface, or to data from which the surface can be derived. More information is available in the programming guide.

1.6.9 *Displacement maps*

The tessellation engine for curved surfaces can dice triangles into micropolygons, the vertex shaders for the vertices can then access into a displacement map and change the location of the points.

1.7 High color depth

We will support a 64 bit color buffer (16:16:16:16), we will support two formats: sRGB64 and a floating point format.. <need to insert format details.

2. Performance

The basic performance is:

R400 MHz fill rate bi-linear equiv peak tri/sec

	MHz	Fill rate	Bi-linear texture fetches	Peak tri/sec
R400	400	3.2 gigapixel	6.4 Billion	400 Million
RV400	500	2.0	4.0	500 Million
R450	500	4.0	8.0	500 Million

Under normal conditions, and when not further limited by memory bandwidth we expect to be > 75% efficient.

3. Schedule

		Tapeout	Samples	Production
R400		July, 2002	Oct, 2002	Dec, 2002
RV400		Nov, 2002	Jan, 2003	March, 2003
R450		Jan, 2003	April 2003	May 2003

4. Process

At the moment this looks like an easy choice: .13 will be in production for over a year, and .10 does not show up until the very end of 2002 according to the TSMC and UMC roadmaps.

We will probably want to be in a flip chip packaging approach to meet power distribution goals. With the 256 bit bus we will have at least 600 signal I/O's (404 in memory). We may be as much as 10A at 1V for average power, which will require very good power distribution, area bond flip chip is probably the only option.

5. General Chip operation

5.1 Unified Shader

The unified shader is a simd/vector engine that performs the same instructions on four sets of four (16 total) elements. For pixel shader operations the elements are pixels with the sets of four required to be 2x2 footprints. For vertex shader operations the sixteen elements are sixteen vertices. The basic element is a 4 value vector – frequently interpreted as x,y,z,w or r,g,b,a.

The user model for the unified shader is composed of a variable number of general purpose registers, a subset of which are usually initialized with data. An ALU can do simple math, conditional moves, and permutations on the registers, and the ability to do a limited number of memory reads using the texture cache. The number of register is variable, and the number of registers required for an operation are specified when the task is submitted to the unified shader. The unified shader will not start the task until there is enough free room for the task's registers.

The unified shader is based on the R300 pixel shader.

5.2 3D Rendering

For 3D rendering data is passed twice through the unified shader- once to transform the vertices and a second time to determine the color of the pixels.

The input to the 3D pipe is expected to be indexed vertex arrays. Linear vertex arrays can easily be supported by the CP generating sequential indices. Inline vertex data is an open issue, I would prefer to write it to memory and then fetch it as a vertex array rather than add a direct path.

The stream of indices is sent to the Primitive Assembly block by the CP. The front of the primitive assembly block maintains the tag for the vertex cache; The vertex cache stores transformed vertices. As misses are detected in the tag, the indices that miss are placed into 16 entry vectors. Each vector contains a state pointer, a pointer to the vertex shader to be used, and the 16 indices to vertices that need to be transformed. When either a vector is filled with 16 entries or a state change happens (so that the next vertex does not share the state and vertex shader with the previous vertex) the vector is issued to one of the "shader" pipelines for transformation. Which of the four shader pipelines it is issued to determined either by some effort of load balancing or a simple round robin. All that is submitted to the pixel pipeline is the state, the vertex program, and the indices. The shader pipeline will fetch the vertex array data through the cache infrastructure that is also used for texture fetches. After the tag the indices (actually now the indices into the vertex cache) are placed into a latency FIFO to hide the latency of transforming the vertices.

The shader pipeline receives the vector of 16 indices from the primitive assembly block. The shader pipeline operates, when rendering pixels, by processing a vector of four 2x2 pixel footprints, a total of 16 pixels. For vertex processing each of the pixels is replaced with a vertex. The vertex program includes information of how many local variables it will need. The rasterizer waits until that many local variables are free, (as each executing thread in the shader pipeline terminates it frees its local variables). With the proposed shader data path the maximum number of local variables per vertex is 256. However this leaves no ability to hide latency, 16 to 32 local variables will probably maximize latency hiding and therefore performance. The vertex shader program can use all the capabilities of the shader pipeline including texture fetches and dependent lookups. At the end of the vertex program, the transformed coordinates must be output. One output will be the x, y, z, w position which we be stored in the position cache of the vertex cache. The vertex program may also output a number of parameter values (colors, texture coordinates, other interpolated inputs into the pixel shader). The parameter values must be output as a multiple of four 128 bit words, as the parameter cache is designed for this.

The primitive assembly block reads the indices back out of the latency FIFO and accesses the position cache portion of the vertex cache. It assembles the vertices into primitives (lines, triangles, rectangles, quads?, points, ?). Baricentric values are assigned to the vertices, and will be used later in the rasterizer to interpolate the parameters. The parameters are not accessed by the primitive assembly logic, which only works from the position data. The primitive is clipped against both the viewing volume as well as user clip planes, with fractional baricentric coordinates assigned to the clipped primitive sections. The primitive goes through the perspective divide and the viewport transform. The resulting screen space primitive is setup (plane equations for 1/W, Z, and the baricentric coordinates). The resulting primitive data, including the indices back into the parameter portion of the vertex cache are broadcast to the four pipes. The final time that an index is output that access the oldest vertex cache line, a token is also sent. When all of the four pipelines return the token the primitive assembly block can free that cacheline and allow it to be used for a new vector of vertices. The performance goal in the primitive assembly block is a triangle every two clocks. An alternative option is for the vertex shader to generate screen coordinates and clip codes. If a primitive needs to be clipped, which can not be determined until primitive assembly, then the vertices are reverse transformed back into clip space by logic in the primitive assembly block, clipped, and then transformed back into screen space.

To help meet marketing BS numbers we can look into doing backface culling at a rate of one triangle per clock. This will boost us to peak bs number of 500 million triangles per second.

Each pipe has a FIFO in front of the rasterizer to load balance. Each pipe will handle 16x16 tiles of the screen which are interleaved between the pipes. To maximize the effective size of the FIFO we will probably cull the triangle list before the FIFO. The rasterizer will request the parameter data from the parameter cache for the primitives. A small latency hiding FIFO will hide the latency of the access to the parameter cache. The parameter cache is 512 bits wide, and the interfaces from the parameter cache to the rasterizer are 128 bits wide, this allows the parameter cache to output one pipelines request per clock, which is serialized over four clocks, keeping all four interfaces busy. The rasterizer keeps a small cache of three to four vertices, this allow only the new parameter to be fetched when adjacent triangles are processed. The parameter cache interface imposes a second performance limits, in the worst case each polygon covers all four pipelines and there are no vertices shared from triangle to triangle. In this case the peak performance is (500 MHz / (4 pipelines * 3 vertices) = (500/12) = 41.6 million triangles per second. In the best case triangles are perfectly stripped and never cross over pipeline boundaries. In this case the peak performance (If we ignore the setup limit) is 500 million triangles per second. As a practical manner we should be able to approach the setup limit of 250 Million triangles per second.

The rasterizer also contains a portion of the hierarchical Z memory. We are looking into moving this into a cache based approach, but that is far for certain at this point. We would like to be able to do hierarchical z culling at a speed in excess of 64 pixel per clock per pipelines (256 pixels per clock total). We are also going to consider some of the improved latency hierarchical Z options to improve culling efficiency.

The rasterizer will generate four pixels per clock if there are no more than eight interpolated parameters. The rasterizer generates vectors of four 2x2 footprints (16 pixels). Each 2x2 footprint must be screen aligned and from the same triangle (with a single shared z slope). The four footprints only need to share the same state and shader program.

Before starting the processing of a vector the rasterizer (which includes the sequencer for the shader pipeline) checks to make sure that there are enough free registers in the shader pipeline for the pixel shader program. If not, it stalls until there are enough. The rasterizer also needs to arbitrate between the three streams of vectors to be shaded: the vertex stream, the pixel stream, and the real time stream. I think it will be sufficient for the real time stream to have priority over the vertex stream which has priority over the pixel stream. This will meet the real-time demands, and keep the vertex cache filled.

The vector is then processed by the shader pipeline. We will probably support up to eight sequentially dependent texture fetches. (to use the R300 terminology, eight clauses). 16 (8?) textures are supported, but each texture can be accessed multiple times by a single pixel shader which can provide a different address each time. This is especially useful for complex filters.

The output of the pixel shader is the final color of the fragment. The pixel shader may also replace the Z value. Fog and stippling must be done in the pixel shader program.

The render backend does the z compare, stencil operation and color alpha blend.

The texture fetch path has a number of design options. One option is an approach where the local, multiported, texture cache is small (1 to 4 KB), and contains uncompressed color in a canonical format (32 bits per pixel) and uses a 4x2 or 4x4 cacheline. This is backed up by a large (>16KB) L2 cache which also stored uncompressed 8x8 cachelines. The decompression logic lives between the memory controller and the L2 cache.

An alternative design uses the L2 cache to contain data in memory format (compressed) which is decompressed as needed to fulfill L1 texture cache misses. This will increase the effective size of the L2. The L2 cache is distributed, with 1/4 of it residing in each memory controller. The Texture decompression logic can either be located in each shader pipeline, or exist as a shared block(s) that receive data from all four memory controller and send the decompressed 4x4 cachelines to each shader pipeline. The unified decompression block will result in better performance, and possibly less area, at the cost of some of the scalability.

Assuming that we chose the L2 in memory controller and the unified decompression logic, the texture path would work as follows:

In a four pipeline design there are two texture decompression blocks, one for the “left” texture units in each shader pipeline, and the second for the “right” texture units. In the two pipeline, lower cost, version of the chip only a single decompression pipeline is used, serving the left and right texture units.

The L1 texture cache receives a texture request from its shader pipeline. The usual tag and latency FIFO is used to generate the misses. These are sent to the shared texture decompression block, which looks up the texture to find the physical address and then sends the request to the L2 cache in the memory controller. The L2 also has a latency FIFO and tag, and will return the data in order (but there is no order guaranteed between the data returning from each L2). The decompression block has a buffer which is used to place the data from the memory controllers back in order. The decompression logic decompresses the texture and returns, in order, the 4x4 cachelines that the L1 caches are requesting. Most of the compression techniques we are considering are based on an 8x8 tile (or 4x4x4), when necessary the decompression logic will decompress an entire 64 pixel tile and only return the requested 16 pixels to the L1 cache. This will tend to increase the bandwidth between the decompression logic and the L2 cache as 8x8 blocks are repeatedly requested to provide different 4x4 subtiles to the L1. The L2 cache will prevent the repeated reads from going to memory, and we will probably implement an “L0” style cache in front of the L2 to also catch the redundant requests.

Each memory controller will have two 64 bit read return buses, one to each of the two decompression blocks, each decompression blocks drives a separate 128 bit bus to each of the four shader pipelines. This will tend to have better utilization and load balancing than having the memory controller drive a 32 bit bus to the decompression logic in each shader pipeline. While the total number of wires is similar (128 bits per memory controller, 128 bits into each texture cache) we are less likely to leave the texture pipes starved when there is some imbalance.

5.3 Real Time Rendering

The real time rendering interface allows primitives to be inserted into the rendering pipeline at a very late stage, therefore providing very low latency. The expected use is for scale blits timed by the display refresh, this suggests a small number of large primitives. We take advantage of this to simplify hardware by forcing the interface to be post setup, a real-time primitive needs to be transformed and setup by software.

Real time primitives also do not have access to the state management hardware used by non-real-time 3D commands. A single set of state registers, some constant registers, and one full parameter set is available. The real-time command stream will generally need to wait for the current real-time drawing operation to complete before it can start the next real-time command. The driver can statically allocate some of the physical constant registers to the real-time stream, these are not available to the RBBM for renaming use, and are written by the real-time command stream, and read by the 3D pipe at the direct physical addresses. There are two options for the parameter memory. The parameter memory is not visible to non-real-time commands, for normal operation it is entirely managed by hardware. For real time rendering there will be dedicated space for three vertices, each with sixteen 128 bit interpolants. If the real-time primitive requires more than eight interpolants there will only be enough room for one primitive at a time, even if they need the same state and constants, if less than eight interpolants are needed then there is room to manually double buffer the interpolants, and allow pipelining of primitives. The real time command stream will still need to manually check that the pipeline has finished with the previous primitive, before writing new data to the parameter memory for the next primitive, while the pipeline works on the current primitive.

For example, the a drawing command in a real-time command buffer might look like this:

```
Wait_for_realtime_pipe_idle                 // make sure no real-time command is in the pipeline
Write state reg m in context 7 with data                      // set  rendering state for command
Write state reg m in context 7 with data                      // set  rendering state for command
Write state reg m in context 7 with data                      // set  rendering state for command
Write state reg m in context 7 with data                      // set  rendering state for command
Write const reg at physical address k                         // write constant register
Write const reg at physical address k+1                       // write constant register
Write const reg at physical address k+2                       // etc.
Write vertex 0, parameter 0, in real time parameter store
Write vertex 1, parameter 0, in real time parameter store
Write vertex 2, parameter 0, in real time parameter store
Write setup primitive to primitive assembly (scan converter)
Write initiator register, tag command with 0
Write vertex 0, parameter 8, in real time parameter store
Write vertex 1, parameter 8, in real time parameter store
Write vertex 2, parameter 8, in real time parameter store
Write setup primitive to primitive assembly (scan converter) // this assumes we double buffer the primitive registers
Write initiator register, tag command with 1
Wait_for_realtime_command_0_not_in_pipe
Write vertex 0, parameter 0, in real time parameter store
Write vertex 1, parameter 0, in real time parameter store
Write vertex 2, parameter 0, in real time parameter store
Write setup primitive to primitive assembly (scan converter)
Write initiator register, tag command with 0
Wait_for_realtime_command_1_not_in_pipe
Write vertex 0, parameter 8, in real time parameter store
Write vertex 1, parameter 8, in real time parameter store
Write vertex 2, parameter 8, in real time parameter store
Write setup primitive to primitive assembly (scan converter) // this assumes we double buffer the primitive registers
Write initiator register, tag command with 1
Wait_for_realtime_command_0_not_in_pipe
```

5.4 State Management

State management differs from previous ATI chips.

There are eight sets of state registers in the chip. Each pixel or triangle is tagged with which state it is supposed to use. Most of this is hidden from the programmer by the RBBM, which implements the in-order semantics that are normally used. States 0 to 6 are managed by the RBBM for high performance 3D/2D/video rendering. State 7 is reserved for real time commands, and the real time command stream must ensure that the state is not changed while the pipeline is active.

Each register is therefore mapped in to the register space nine times: once for the current state, plus eight additional times to provide access to all existing state. This is only true for the normal pipeline state registers, the constant registers used by the pixel/vertex shaders are handled by a separate, related mechanism.

There are two options for the update of the state registers. The first option is to implement a broadside state copy, which copies the contents of the previous current state to the new current state before the first state write happens to the new state. This is somewhat costly in hardware. The second option is for the state updates done by the driver to be “complete”, write the minimum set of state registers that completely defines the new rendering state, this avoids the need for the hw broadside copy.

The constant registers are implemented using a renaming scheme that avoids the need to do a broadside copy when changing state. It also does not use storage for each state, when two state contexts have the same value in the same register, the renaming logic points them at the same physical register.

Since the registers that are most frequently changed are located in the constant memory of the R400 (vertex array pointers, and texture pointers) we may wish to separate updates to the constant registers from general state register updates.

5.5 Bad Data

Bad data can exist for a number of reasons. When a vertex shader does an access to an address which is not permitted (or does not exist) we need a way to avoid hanging, and make debugging possible; A similar issue exists for pixel shaders that do bad texture accesses.

We currently handle a limited form of this: a triangle than contain a vertex which contains (or generates) a NaN or INF is not drawn, it is simply culled at setup.

We will extend this as follows:
For a vertex fetch that goes out of range (or times out) a flag in the vertex is set which will cause that vertex to be treated as if it contained a NaN. A debugging flag will also be set, and if we can find an easy way to do it, the index of the offending vertex will also be stored.

For a texture fetch a similar strategy will apply: A bad access will set a flag that will cause the pixel to be dropped. The debugging mode will force the pixel to pass the Z test, and override the color output from the pixelshader with an ugly shade of green.

5.6 Display operation

The display must be able to display from microtiled surfaces and overlays. This will generally force us to adopt line buffers.

The display should support at least two outputs, ideally we will be able to support two high resolution outputs and a low resolution output (TV out)

We will drop support for overlay scaling, and therefore supporting an overlay on all displays becomes affordable, fixing a “bug” that our current dual display products suffer from.

The memory for the line buffers is shared with the L2 texture cache. This allows use a memory size that is closer the maximum requirement of either function, instead of the sum of the maximum requirement.

The maximum resolution color format is 64 bit color for the primary surface and 32 bit color for the overlay
For two 2560 pixel wide line buffers we need

2560 pixels	2560	
two lines	2	
two displays	2	
96 bits of color	96	(32 overlay + 64 bits primary)
total bits	960K bits, 120 Kbytes	

The L2 memory will probably be 128Kbytes, which will leave only 8KB for the texture L2 cache when driving the above display. However, the above case is driving two multi-megapixel displays with the worst case color depth. It works, but 3D performance suffers.

A slightly more normal case might be two 1600x1200 displays, with the same color depth:

2560 pixels	1600	
two lines	2	
two displays	2	
96 bits of color	96	(32 overlay + 64 bits primary)
total bits	600K bits	

which leaves >54Kbytes for the L2 Cache

A benchmark case, one display no overlay, 32 bit color:

2560 pixels	1280	
two lines	2	
two displays	1	
96 bits of color	32	(32 overlay + 64 bits primary)
total bits	81K bits	

which leaves >110Kbytes for the L2 cache.

The line buffers are two scan lines high:

At beginning of an even scan line (scan line 0) 1/2 of line buffer is filled.
The upper half of the buffer is read out, at the same time as the second half of the buffer is filled. When the scanout reaches the midpoint 3/4 of the buffer is filled. When the scanout reaches the end of the scan the buffer is filled. The speed at which the buffer is filled must be greater than 1/2 of the rate at which the buffer is scanned out.

For the odd scanlines, the buffer is completely filled at the start of scanout (as a result of the even scanline finishing properly). As the lower half is scanned out, reads are issued to fetch the data for the next pairs of scanlines. At the end of the odd scan line, the buffer is expected to contain half of the data for the next scan line.

Another way of looking at this is as follows:

At the beginning of the odd scan line the scan buffer is filled. As each word is read from the buffer and sent to the display logic, a request is made to the memory controller to fill in the data. It is not necessary for all the data for the next scan line pair to be fetched by the time that the scan line reaches the end, the real requirement is for the last word in the scan line buffer to get there just before it is read, at the end of the next even scan line.

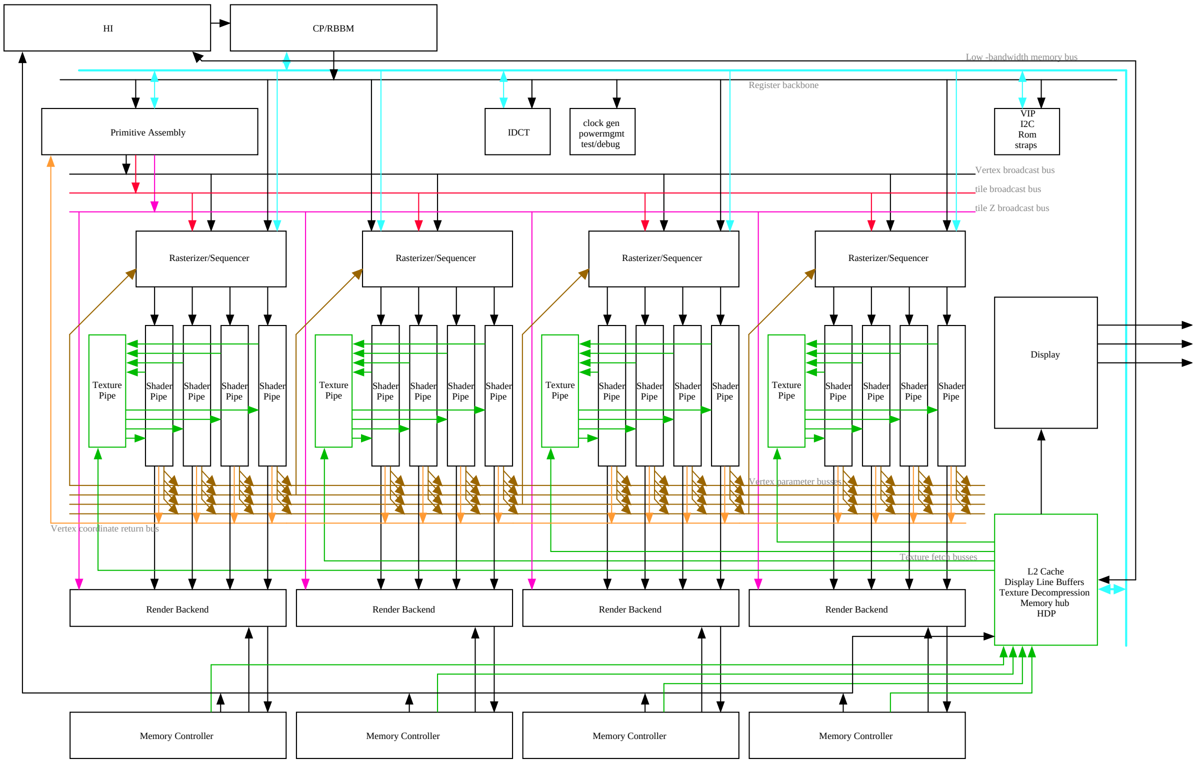
The display lives mostly in the core clock domain. There is a FIFO per pixel clock that crosses into the DAC/TMDS clock domain.

If we are able to implement the time interleaved display block then the display will merge and color convert two pixels per clock in the core clock domain. Whichever display FIFO is closest to empty will get the priority to be filled in the next time slot. The sum of the pixel clocks (display0, display 1, tvout) must be less than 2x the core clock. We should be able to cheat slightly and use some of the horizontal retrace time to fill the display fifo's, this will relax slightly the 2x core clock limit.

Since the 3D pipe is capable of real-time events, such as display triggered scale-blits, we may wish to reconsider the location of several operations that are currently in the display. TV out scaling, and ratiometric scaling for LCD panels may be more cheaply implemented using the 3D pipe instead of dedicated hardware.

6. Block Diagram

R400 Top Level Block Diagram



7. Blocks

HBIU – host bus interface unit
CP – control processor
RBBM – register interface manager
CLK – clock generator
TC – test controller
VIP – video input port

ROM – boot rom
I2C – I2C interface
DU – Display
MH – Memory Hub
HDP – Host Data Path
IDCT – Mpeg decoder
PA – Primitive Assembly
TD – Texture Decompression
RE – Raster Engine
SP – Shader Pipe
TP – Texture Pipe
RB – Render Backend
MC – Memory Controller
The blocks are combined into a smaller number of blocks for layout:

Layout block	subblocks	Instances R400/450	Instances RV400	Notes
HI	HI	1	1	
CP	CP RBBM CLK Reset	1	1	
Misc	VIP ROM I2C TC	1	1	
DU	DU	1	1	Display
TD	TD MH HDP	1	1	L:2 Cache
PA	PA	1	1	
RE	RE	4	2	
SP	SP	16	8	
TP	TP	4	2	
RB	RB	4	2	
MC	MC	4	2	

8. Block descriptions

8.1 HBIU – host bus interface unit

The HBIU interfaces the graphics chip to the system AGP bus.

8.1.1 *Description*

The HBIU implements the following buses:
PCI slave
PCI master
AGP fast writes
AGP reads
AGP writes

64 bit support?

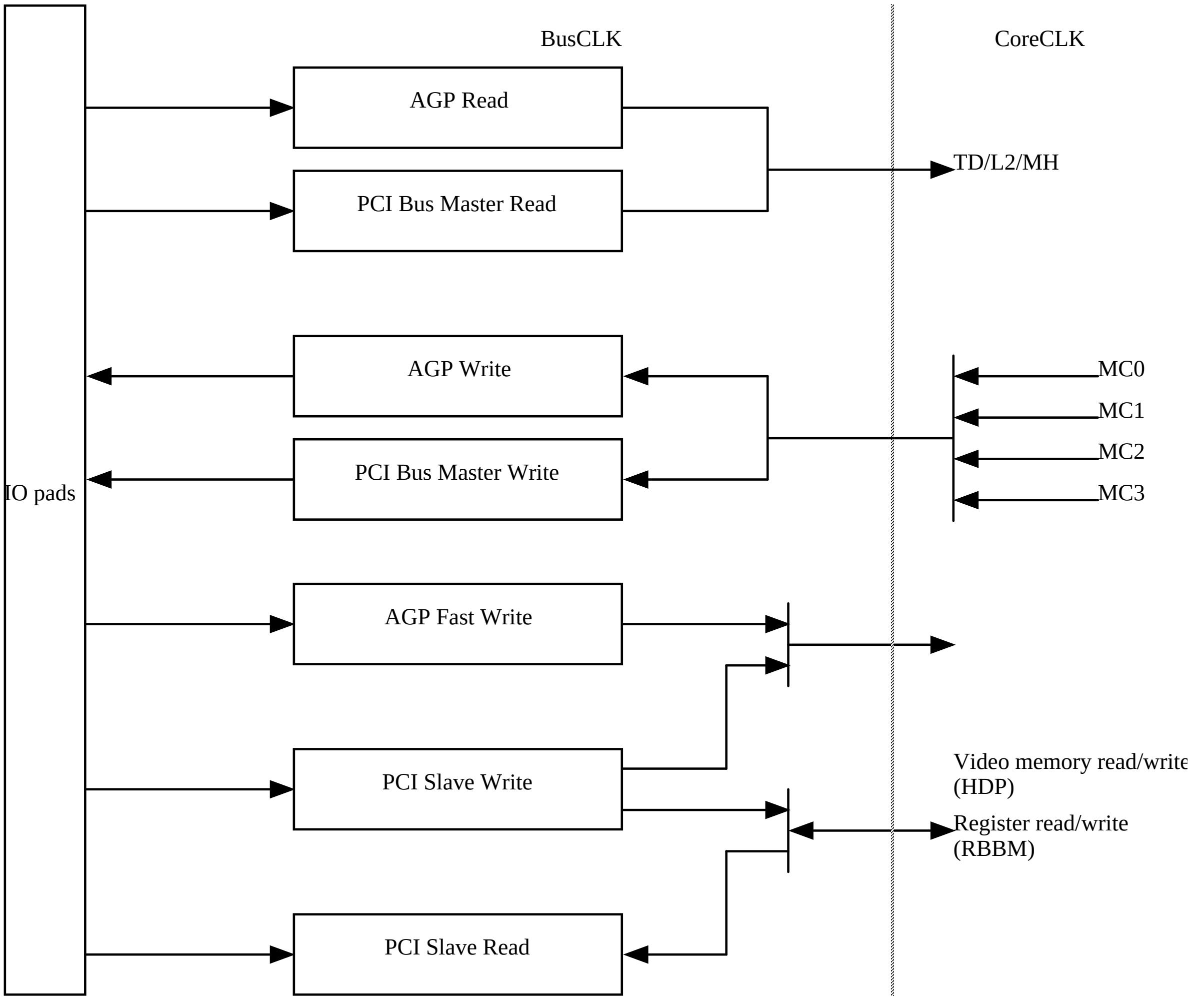
8.1.2 *Major interfaces*

The following busses connect the HBIU to the rest of the chip:

Bus	Chip client	Bus client	Description

Host register	CP/RBBM	PCI Slave	CPU reads and writes to chip registers
Host Data	HDP	PCI Slave AGP fast write	CPU reads and writes to video memory
AIC Write	MC	PCI Master (writes) AGP writes	Primarily blits to system memory, and control semaphore writes
AIC Read	TD/MH	PCI Master (reads) AGP reads	CP PM4 reads PA index reads State/vertex program loading Vertex loads AGP texture
Oddites for VGA			

8.1.3 *Block diagram*



8.2 CP – control processor

8.2.1 *Description*

The control processor executes the pm4 display list from memory, driving the operation of the rest of the chip. It also implements the real-time event commands.

Currently the CP is based on a custom processor, which has a very limited instruction set and is really only capable of executing the existing program. It is not expected to be capable of doing the translation of 2D packets to the preferred hardware interface, or be able to implement the real time commands.

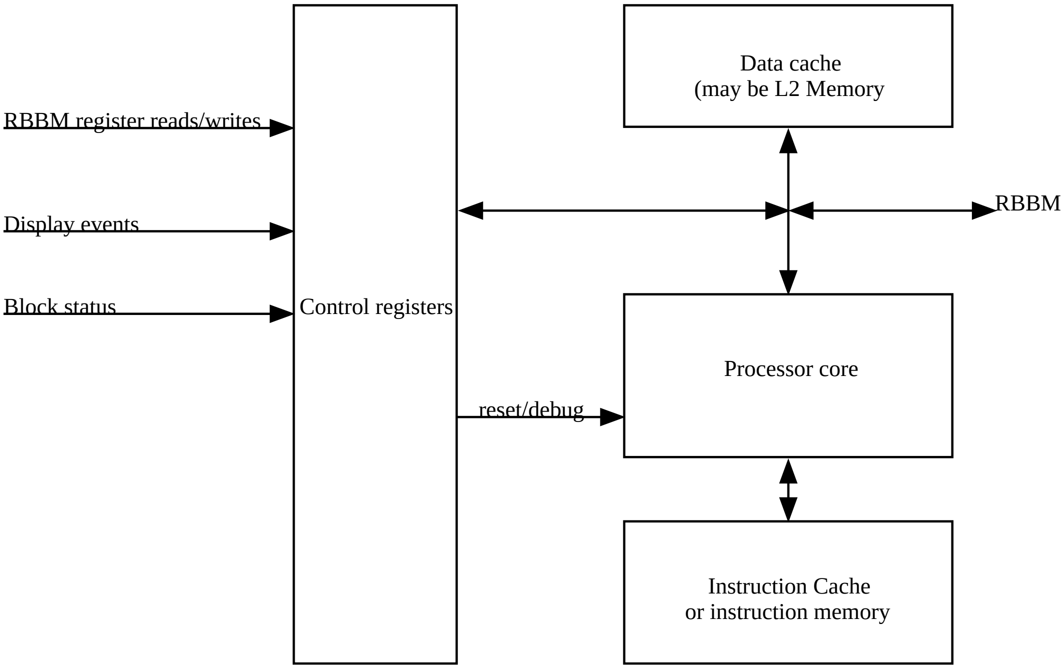
An alternative is to base the CP on a more generic RISC processor. It appears that this will save area, and make it possible to write the CP control program in C. The ARC core, for example, is less than 20K gates.

One key change that enables us to consider a processor core instead of the custom PM4 engine is that data is no longer embedded in the command stream. In the R128 to R300 index, vertex, and host-blit data is embedded in the primary ring buffer and the indirect buffer. In the R400 index data is fetched by a dedicated DMA engine in the PA block, and vertex and host blit data is fetched through the texture cache. This allows us to optimized a single path for the data rather than need to optimize both the DMA and PM4 paths. With the CP no longer needing to be able to copy data at 32 bits per instruction (read and write), a less specialized processor can be used.

8.2.2 *Major interfaces*

Bus	Description
RBBM->CP	Register read write, Used for reset and debugging of CP, and access to control registers
CP→RBBM	Register writes, and reads Register access that occur as a result of executing the control program
CP→MH	Memory reads and writes. Read PM4 buffers, write semaphores to communicate with driver
Display→CP	Source of real time events to trigger real time commands, also delays in command queue based on display status. Current scan line is most common type of data
All block→CP	Blocks status. Used for wait for idle and power down

8.2.3 *Block diagram*



8.3 RBBM – register interface manager

8.3.1 *Description*

The RBBM of the R400 is vastly simplified compared to previous versions.
The key differences are:

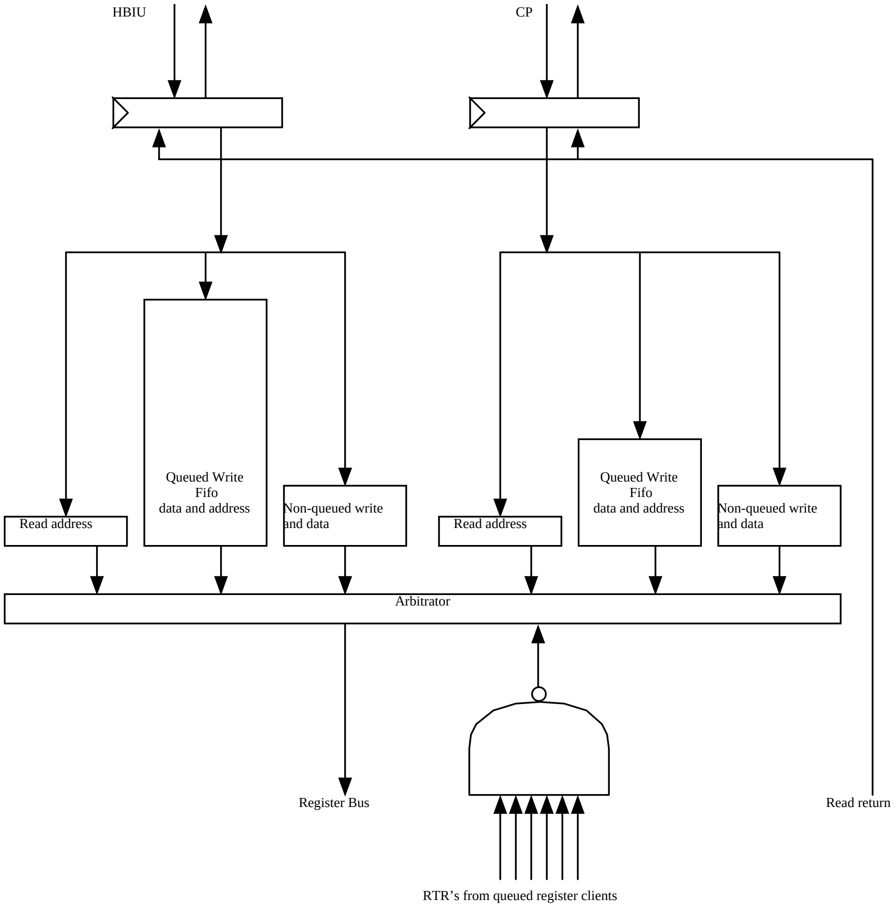
1) A much simpler register decoding scheme that does not need the RBBM to be aware of autoreg files.
2) A simpler register bus protocol that (for most registers) does not involve any feedback signals to the RBBM
3) Support for simple pipelining of the register bus to meet timing goals.
4) Much of the synchronization logic that was in the RBBM is now the domain of the CP, this means that bypassing the CP is not a viable production driver mode, but it really is not viable now.
5) Power Saving needs some adjustment (since the RBBM is no longer aware of when a block is activated.
6) All register bus connections are now single cycle, register to register which will simplify timing.

However the management of state changes has been moved from the blocks in the 3D pipe to the RBBM. The RBBM detects when a state block is no longer in use, tracks the blocks that are not is use, and allocates them to new primitives as needed.

8.3.2 *Major interfaces*

Bus	Description
HBIU→RBBM	Register read/write
CP→RBBM	Register writes resulting from interpretation of command packets
RBBM-register bus	The purpose of the block
RBBM→CLK	Power management
RBBM→all	Soft/hard reset

8.3.3 Block diagram


HBIU
CP
Queued Write Fifo data and address
Read address
Non-queued write and data
Queued Write Fifo data and address
Read address
Non-queued write and data
Arbitrator
Register Bus
Read return
RTR's from queued register clients

8.3.4 RBBM operation

This is copied from the current RBBM spec, at some point most of it will be moved back there.

The RBBM has merges register writes and reads from the HBIU and the CP and broadcasts them to the rest of the blocks in the chip. That is all it needs to do.

Registers can either be queued or un-queued. In general queued register writes are initiator registers, or order critical state registers. The RBBM distinguishes between the two types of registers by their address only. The upper ?Kbytes of the register space are queued registers, the remainder is un-queued.

Both the CP and the host can generate both types of register writes.

Un-queued register writes can and will pass queued registers writes. If it is important for un-queued register writes to be held off by a queued register write the host or cp must not send the un-queued register write until the host or cp has determined that the queued register write has completed (usually by a spin lock on a semaphore).

Queued registers are maintained in order from the viewpoint of each originator. I.e. all of the CP's queued writes will complete in order, and all of the hosts will complete in order. There is no ordering between the CP and host- the writes from both clients may become interleaved.

The global register bus is as follows:

Name	Direction	bits	Description

WE	RBBM→	1	Write enable, address and data are valid
Addr[19:2]	RBBM→	18	Register address
Wm[3:0]	RBBM→	4	Register write mask, should be ignored by most clients
Wd[31:0]	RBBM→	32	Data
RE	RBBM→	1	Read Enable, address is valid
Rd[31:0]	→RBBM	32	Read data returned
RRn	→RBBM	1	Read return strobe (active low)

The protocol for a write is simple:
On a rising edge , if WE is high then the data and address is valid.
There are no completion signals, there is no way to abort a write.
Handshake signals for queued registers will be described later and are separate from the register bus.

The read protocol is somewhat more complex.
A read request is sent out when RE is high. The address holds the address of the read request.
RE and addr will only be valid for one clock cycle.

Some number of clock cycles later the RBBM will receive the return data back, when RR is low.
The read return "bus" (Rd and RRn) is the logical AND of all the clients that can respond to a read request.
All clients but the client that is responding to the read request drive a logical 1 on the bus. The wiring of the read return bus is a tree of point to point connections, and each node one or more sub-busses are AND'd together, registered, and driven on to the RBBM. This is the same as an OR tree, but the signal is inverted. Since a read return cycle is surrounded by idle cycles the only critical transition is high to low for the Rd signals (possible timing help, at the cost of needing to tell static timing to ignore low to high transitions).

Only one read is outstanding at any time. Reads will pass queued writes, (should/can they pass all writes?)
The RBBM will timeout on a read after 64? clocks. It is critical that no client respond latter than 64 clock as the RBBM may timeout on the read, issue another and interpret the very late response of the first read as the second read.

If a read times out the RBBM will return dummy data (such as '0xDEADBEEF') to the requestor and mark in a debug register than an error happened.

Both queued and non-queued register writes are broadcast on the same bus. To implement the queued registers the RBBM looks at the status of all of the RTR signals from the clients that contain queued registers. Only if all are high will any queued register be allowed to issue from the RBBM. Note that these signals are registered on the boundary of the RBBM, and the register bus is also registered. This means that there is at least a two clock latency responding to a RTR signal deasserting. Since the clients will also be registered this means that a client will receive four or more queued register writes after asking them to stop. It is the clients responsibility to have enough buffering so that no register writes are lost.

8.4 CLK – clock generator

8.4.1 *Description*

The clock generator block generates the many clocks used by the R400:

Clock	speed range	
AGP	133 to 533	AGP clock
Sclk	33to 500 Mhz	core clock
Mclk	33 to 500	memory clock
P0clk	10 to 450	pixel clock for primary display (5x faster for TMDS//LVDS)
P1clk	10 to 450	pixel clock for secondary display (5x faster for TMDS//LVDS)
Tvclk		pixel clock for tvout

8.4.2 *Major interfaces*

Bus	Description

RBBM→CLK	control

8.4.3 *Block diagram*

8.5 TC – test controller

8.5.1 *Description*

8.5.2 *Major interfaces*

8.5.3 *Block diagram*

8.6 VIP – Video input port

8.6.1 *Description*

8.6.2 *Major interfaces*

Bus	Description
VIP→MH	DMA transfers
RBBM→VIP	Control

8.6.3 *Block diagram*

8.7 ROM – boot rom

On powerup the graphics chip reads the straps from the rom. The rom is then used for responding to boot rom read requests from the PCI bus.
We will only support serial roms.
The list of supported roms is TBD.

8.7.1 *Description*

8.7.2 *Major interfaces*

Bus	Description
ROM→HBIU	Boot rom read interface
RBBM→ROM	Flash/eeprom boot rom write interface
ROM→chip	Decoded straps

8.7.3 *Block diagram*

8.8 I2C – I2C interface

8.8.1 *Description*

The I2C bus is a 2 wire bus used to communicate with other multimedia devices (such as tv tuners)

8.8.2 *Major interfaces*

Bus	Description
RBBM→I2C	Read/write interface

8.8.3 Block diagram

8.9 DU – Display

The R400 display drives up to three displays: two monitors and a TVOUT.

The chip can have as much as two analog RGB DAC's, two dual channel TMDS outputs, and one dual channel LVDS output.

All support for the scaling overlay is removed. The display supports a non-scaling overlay on each display.

See display operation section above for more details.

8.9.1 Description

Hopefully we have the resources to move to the time interleaved display design.
The following frame buffer formats are supported:
Primary surface:
8bpp index
16bpp 4444,565.555 RGB
32bpp 8888 RGB
64bpp 16:16:16:16, either sRGB or the R400 floating point format

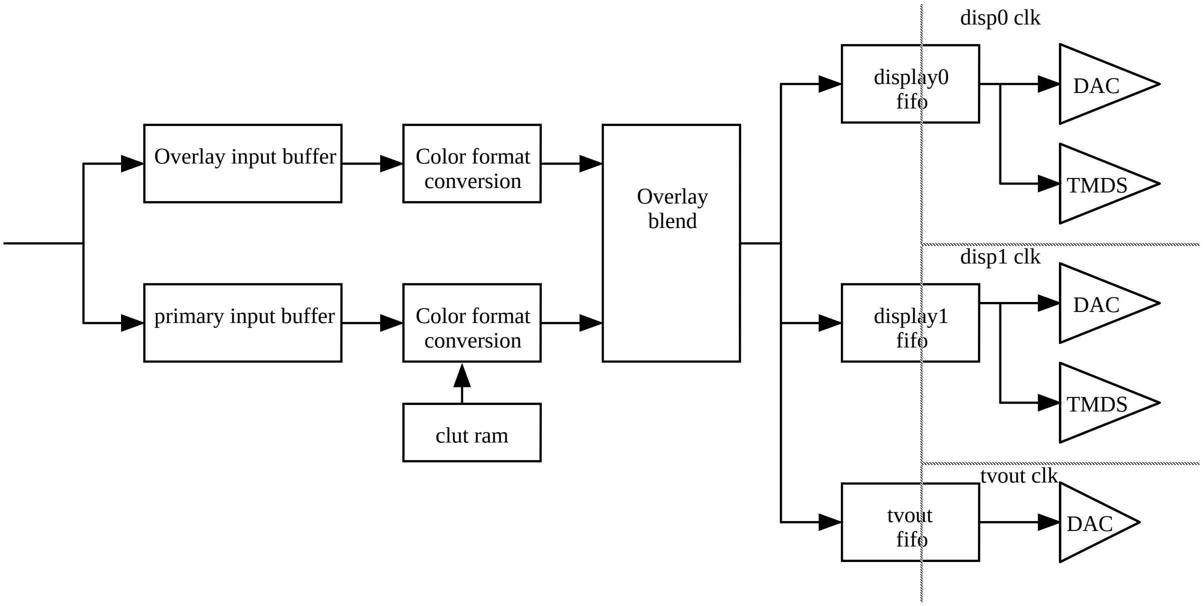
Overlay:
32bpp 8888 RGB
4:2:2 YUYV
the color conversion for the overlay is controlled by a programmable matrix, so the choice of color space is arbitrary.

The maximum display pixel clock is greater than 400 MHz.
If we build the time interleaved design, then the maximum number of display pixels will be 2x the core clock speed. This will be divided among the two displays and tvout.

8.9.2 Major interfaces

Bus	Description
TD→DU	Memory read interface
RBBM→DU	Register writes/reads
DU→CP	Synchronization information

8.9.3 *Block diagram*



8.10 MH – Memory Hub

8.10.1 *Description*

The memory hub acts as a switch between the many small clients of the memory controllers, and the two or four memory controllers. This allows most blocks to not have any dependencies on the number of memory controllers.

8.10.2 *Major interfaces*

The memory hub has a 32 bit read and a 32 bit write bus to each of the memory controllers. If we co-locate the MH and the L2 cache in the texture decompression block, then the 128 bit read return bus from the MC to the L2 cache can be used to return read data to the MH instead of a private bus.

The clients of the MH:

HDP	
CP	
VIP	
PA-	index buffer reading
RE-	Vertex/pixel program loads, hierarchical Z
IDCT	
?	

8.10.3 *Block diagram*

8.11 HDP – Host Data Path

The host data path allow the host cpu to access video memory. It provides eight “surfaces” that provide endian and tiling translation, making the target area of memory look like a linear surface in the processors native endian.

The HDP also implements most of the legacy VGA functionality.

8.11.1 Description

8.11.2 Major interfaces

Bus	Description
HI→HDP	Memory read/write requests to HDP
HDP→MH	HDP reads/writes to local memory

8.11.3 Block diagram

8.12 IDCT – Mpeg decoder

8.12.1 Description

The R400 uses the same implementation of IDCT/MPEG as the R300. This block decodes the compressed stream, placing the resulting IDCT data in one buffer in memory, and the motion vectors in another. The 3D pipe is then programmed to read the motion vectors and IDCT data and complete the decoding operation

8.12.2 Major interfaces

Bus	Description
IDCT→MH	Read command stream, write IDCT results and motion vectors

8.12.3 Block diagram

8.13 PA – Primitive Assembly

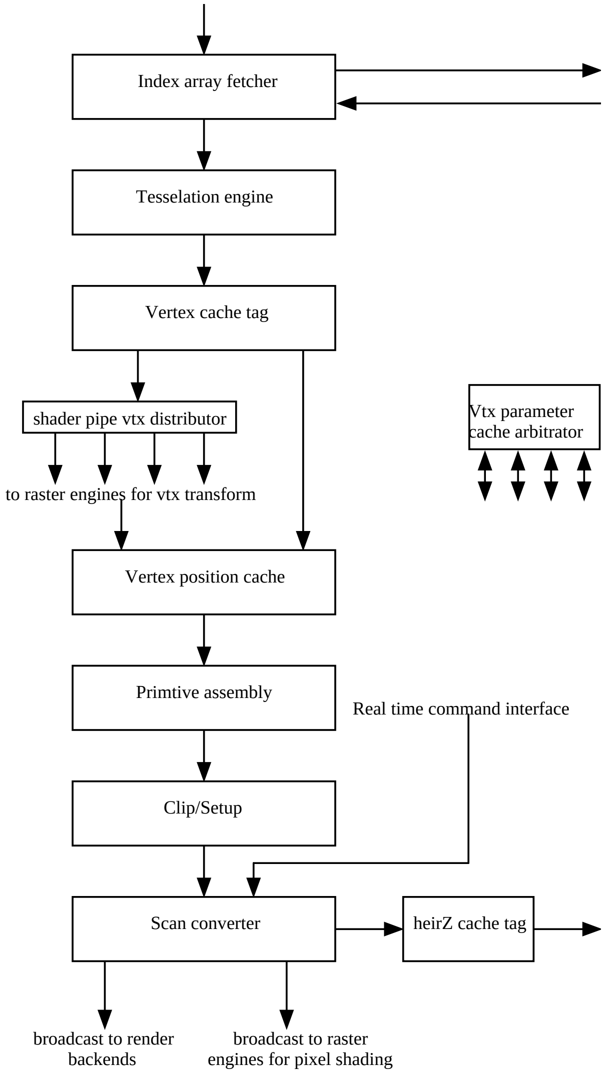
8.13.1 Description

The primitive assembly block fetches or creates the indices to vertices, possibly creating extra vertices with the tesselation engine. It determines which vertices have not been recently seen (and will therefore not be located in the post transform vertex cache), assembles vectors of sixteen vertices than need to be transformed, and submits them to a raster engine/shader pipe set to be transformed. It then receives the transformed vertex position data from the shader pipes. The vertex cache tag also outputs the sequence of cache addresses generated from the incoming indices. The primitive assembly subblock then creates primitives (lines, points, rectangles, triangles) from the vertices. It also implements the line counter for styled lines. The primitives are setup in the setup/clip block, but first clipped to the view frustum and optionally the user clip planes. The scan converted does a course walk of the primitive using an 8x8 grid. The scan converted also determines when the hierarchical Z data needed for culling will not be in the local hierarchical z cache in each rasterizer and makes the needed memory read requests. An arbitrator, centrally located in the PA block arbitrates each rasterizers reads from the post transform vertex parameter cache, which is distributed among the shader pipes.

8.13.2 Major interfaces

Bus	Description
RBBM→PA	State changes, and initiator register writes
PA→MH	Index fetch path
PA→RE	Vertex transform packets
PA→MH	Hierarchical Z read request
PA→RBn	Coverage mask, position, and Z slope
PA→SPn	8x8 tiles to be rasterized

8.13.3 *Block diagram*


Index array fetcher
Tesselation engine
Vertex cache tag
shader pipe vtx distributor
to raster engines for vtx transform
Vtx parameter cache arbitrator
Vertex position cache
Primtive assembly
Real time command interface
Clip/Setup
Scan converter
heirZ cache tag
broadcast to render backends
broadcast to raster engines for pixel shading

8.14 TD – Texture Decompression

8.14.1 *Description*

The texture decompression block converts the memory texture formats into the uncompressed texture formats supported by the texture pipes. It consists of the L2 texture cache, the texture decompression logic, a set of output buffers, and the texture addressing logic.

The decompressed formats supported are:

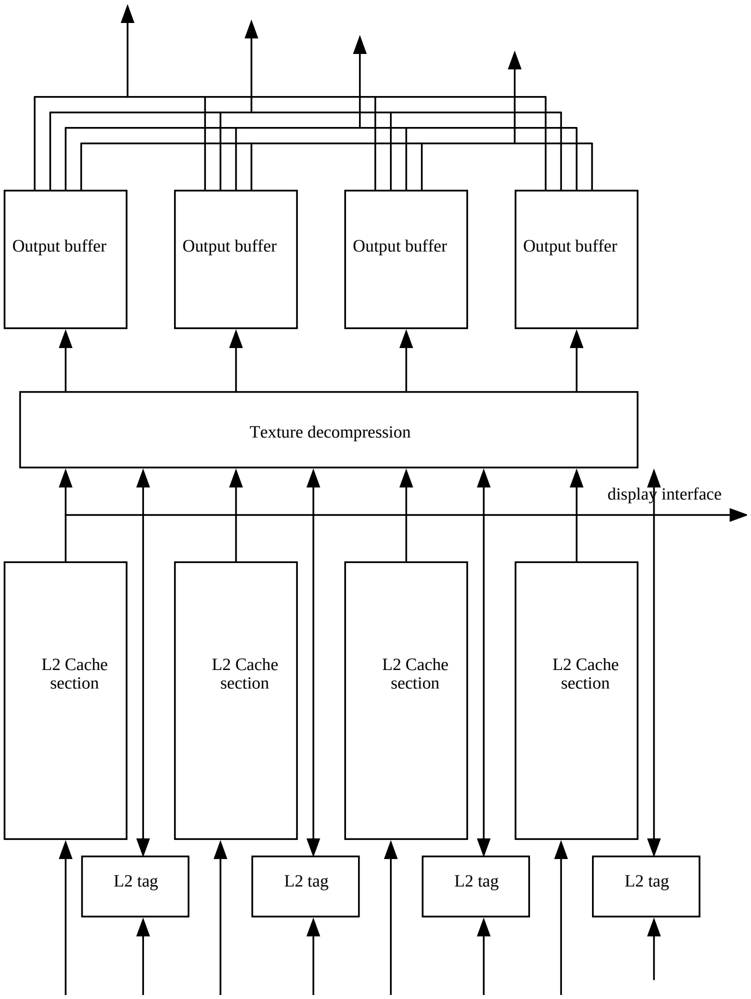
32 bpp (8888) unsigned
32 bpp (8888) signed
32 bpp (16,16) unsigned and signed
32 bpp (32) unsigned and signed

we may also support a 8bpp mono format to improve the performance of shadow buffering.

8.14.2 *Major interfaces*

Bus	Description
TP→TD	Texture requests and returned data
TD→MCn	Memory read requests
TD→Display	Data path for display which uses the L2 cache as its line buffers
MCn→TD	Invalidate snoop bus for cache coherency

8.14.3 *Block diagram*



8.15 RE – Raster Engine

8.15.1 *Description*

The raster engine performs two duties: it does the detail walk of 8x8 tiles of primitives, and it contains the sequencer for the shader pipe.

The shader pipe has the FIFO to allow for balance between the pipelines in the chip, it appears that this FIFO is 64 8x8 tiles deep. Only tiles that are owned by this pipeline are stored in the FIFO, others are immediately rejected. When an 8x8 tile is read out of the FIFO, it is checked against the heir-Z fail data that has arrived in the local heir-Z cache. If the primitive fails, it is rejected and the RB is informed that the tile has been killed.

We are going to support hierarchical Z object culling within the command stream. To support this we have the ability to draw a bounding object, heir-Z test it, but kill it before we rasterize it. The raster engine will receive tiles that are marked indicating that they are part of an occlusion query, and test them against the heir-z memory. All the tiles are rejected, but if any of them pass the heir-Z test then id then a flag is set. When the marker (which is an id) changes, indicating the end of this occlusion query, the RE will signal back to the primitive assembly if the flag was set or not.

If the tile passes the heir-z test we need to ensure that the parameter data needed to interpolate the triangle is either in the local l0 parameter cache, or on its way there. If not a request needs to be made to the arbitrator in the PA to get the needed data/

A FIFO on the output of the HZ cull and parameter cache tag buffers the passing tiles while waiting for the parameter data to arrive at the cache. It also provides buffering so that the rest of the pipeline can stay busy during a long string of tiles that fail the hierarchical Z test.

The next step is a detail walker that generates the coverage mask for each potentially covered 2x2 quad in the 8x8. We may need a path from this result to the render backend to aid in its determination as to what to fetch. The parametric coordinates are calculated, and used to driver the interpolator. We need to be able to do both perspectivly correction interpolation and non-corrected interpolation.

The raster engine breaks the stream of pixels into 4 quad vectors (16 pixels) and will wait until the needed space is available in the shader pipe, and then start the sequencer running the pixel shader program.

The second part of the raster engine is the sequencer.
The sequencer first arbitrates between vectors of 16 vertices that arrive directly from primitive assembly and vectors of 4 quads (16 pixels) that are generated in the raster engine.

The vertex or pixel program specifies how many GPR's it needs to execute. The sequencer will not start the next vector until the needed space is available.

The sequencer is based on the R300 design. It chooses an ALU clause and a texture clause to execute, and execute all of the instructions in a clause before looking for a new clause of the same type. Each vector will have eight texture and eight alu clauses, but clauses do not need to contain instructions. A vector of pixels or vertices ping-pongs along the sequencer FIFO, bouncing from texture reservation station to alu reservation station. A FIFO exists between each reservation stage, holding up vectors until the vector currently occupying a reservation station has left.. A vector at a reservation station can be chosen to execute. The sequencer looks at all eight alu reservation stations to chose a alu clause to execute and all eight texture stations to chose a texture clause to execute. The arbitrator will give priority to clauses/reservation stations closer to the top of the pipeline. It will not execute an alu clause until the texture fetches initiated by the previous texture clause have completed.

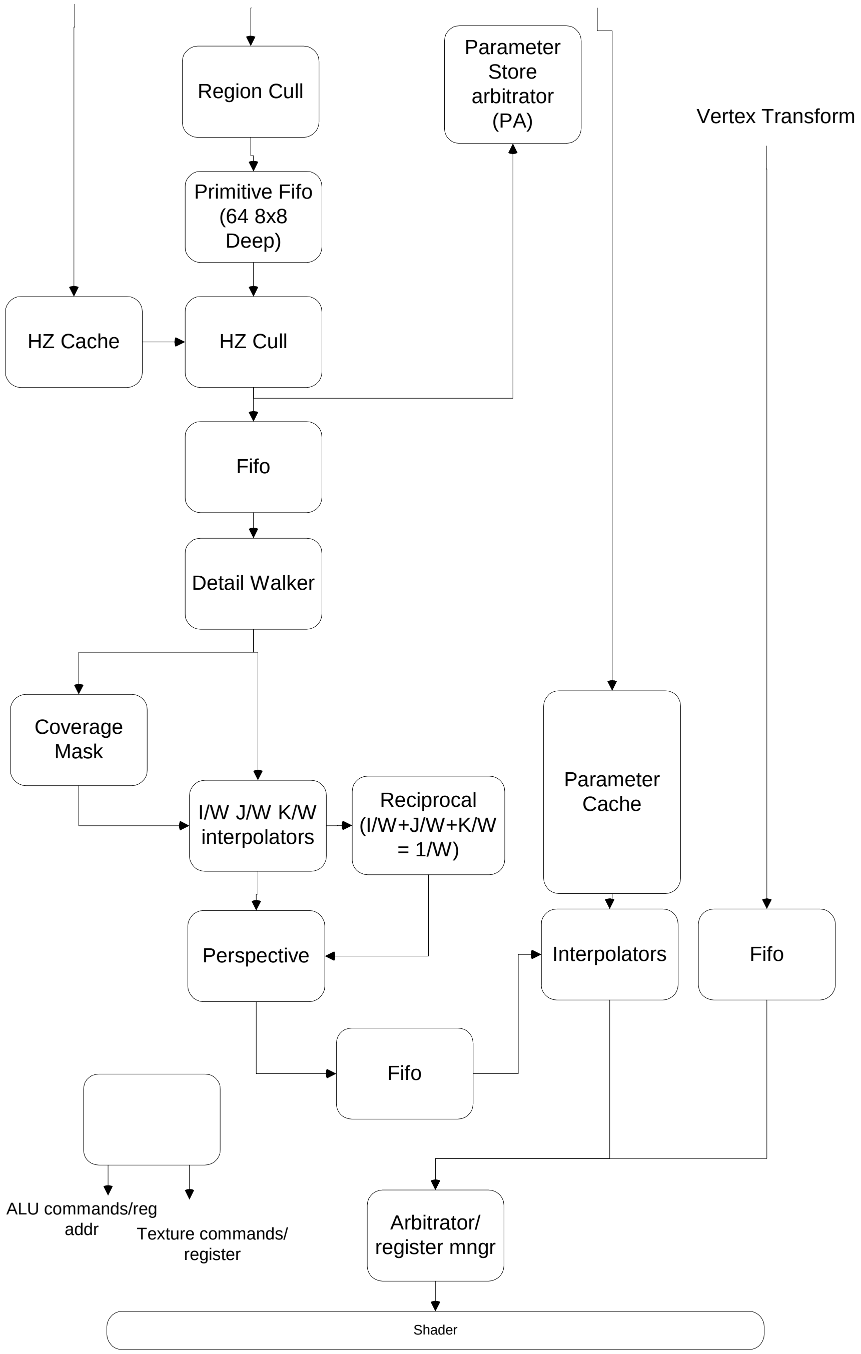
To support the shader pipe the raster engine also contains the shader instruction cache and constant store.

8.15.2 *Major interfaces*

Bus	Description
PA(sc)→RE	Broadcast bus for 8x8 slices of primitives. I,J,K plane equations, front most Z for heir-Z culling, pointer for location of parameter data in vertex parameter cache
MH→RE	Returned hierarchical Z data for local cache.
PA→RE	Parameter request port
SC→RE	Returned parameter data
PA→RE	Requests to transform packets of vertices
RBBM→RE	State register reads/writes
RE→SC	Interpolated parameter data
RE→SC	Instructions, constants, register file addresses
RE→RB	Heir-Z pass/fail information
RE→RB	Sequencing information for availability of pixels
RE→PA	Sequencing interface for returning transformed vertices.
RE→PA	Occlusion query results
MH→RE	Shader I Cache fills

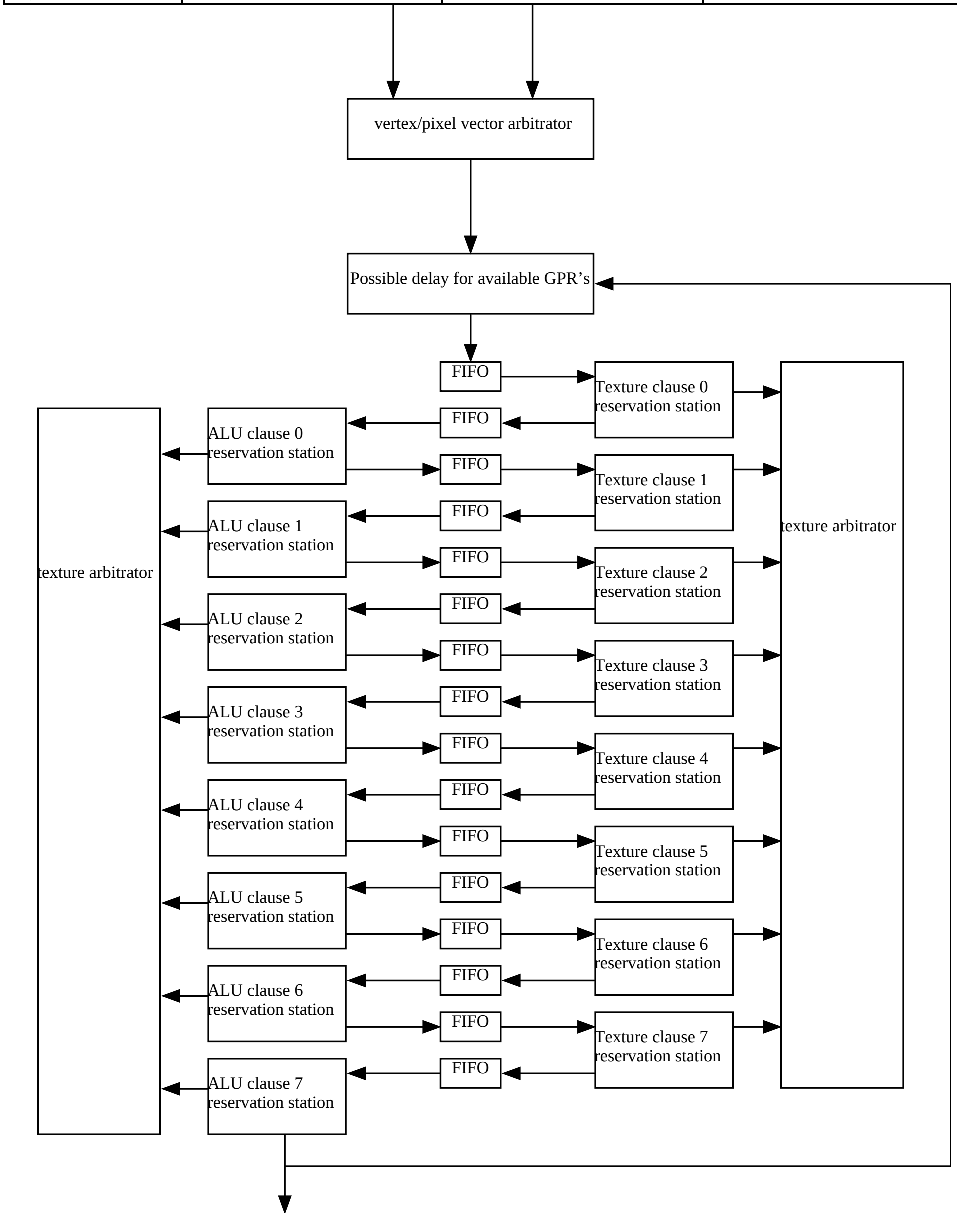
8.15.3 *Block diagram*

8.15.3.1 RE Block diagram


Region Cull
Parameter Store arbitrator (PA)
Vertex Transform
Primitive Fifo (64 8x8 Deep)
HZ Cache
HZ Cull
Fifo
Detail Walker
Coverage Mask
Parameter Cache
I/W J/W K/W interpolators
Reciprocal (I/W+J/W+K/W = 1/W)
Perspective
Interpolators
Fifo
Fifo
ALU commands/reg addr
Texture commands/ register
Arbitrator/ register mngr
Shader

8.15.3.2 RE sequencer

8.15.3.3 RE sequencer arbitrator


vertex/pixel vector arbitrator
Possible delay for available GPR's
FIFO
Texture clause 0 reservation station
Texture clause 1 reservation station
Texture clause 2 reservation station
Texture clause 3 reservation station
Texture clause 4 reservation station
Texture clause 5 reservation station
Texture clause 6 reservation station
Texture clause 7 reservation station
texture arbitrator
ALU clause 0 reservation station
ALU clause 1 reservation station
ALU clause 2 reservation station
ALU clause 3 reservation station
ALU clause 4 reservation station
ALU clause 5 reservation station
ALU clause 6 reservation station
ALU clause 7 reservation station
texture arbitrator

8.16 SP – Shader Pipe

8.16.1 *Description*

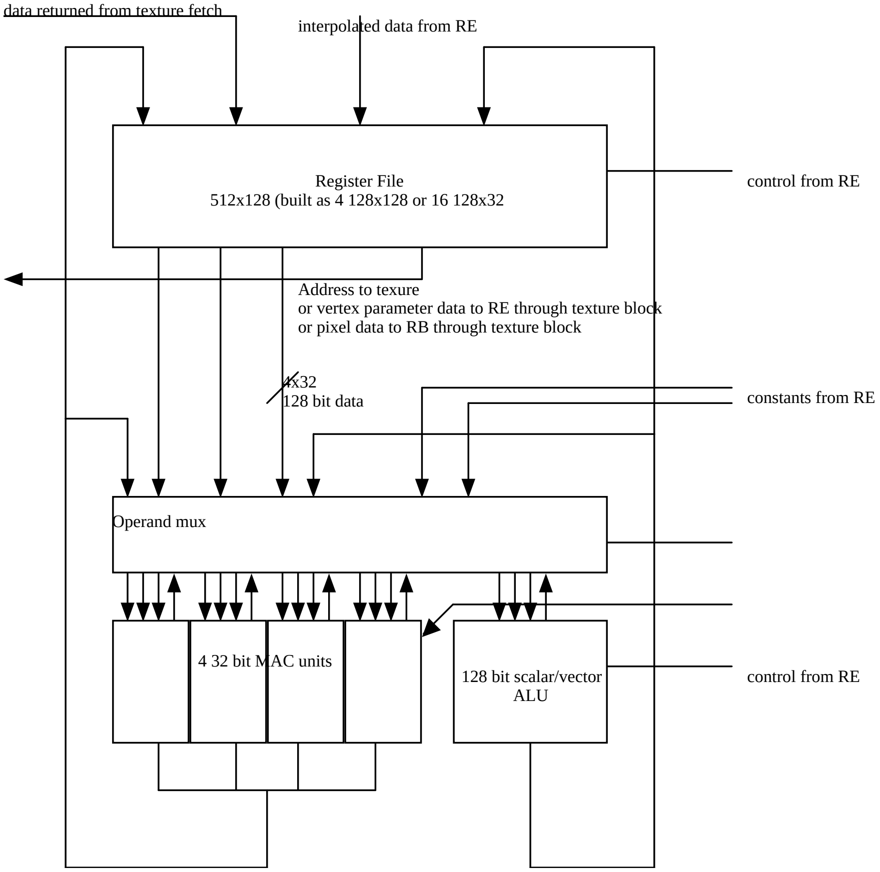
The shader pipe implements the math pipeline of the R400. It has no sequencing/control logic; the control is located in the raster engine.
The shader pipe contains four floating point MAC's, and an

8.16.2 *Major interfaces*

Bus	Description
RE→SP	Interpolated data
RE→SP	Control

SP→TX	Texture requests + vertex parameters + pixels to render backend
TX→SP	Returned texture data
RE→SP	Constants
SP→SP	Local w bus for derivative opcode

8.16.3 Block diagram


data returned from texture fetch
interpolated data from RE
Register File
512x128 (built as 4 128x128 or 16 128x32
control from RE
Address to texure
or vertex parameter data to RE through texture block
or pixel data to RB through texture block
4x32
128 bit data
constants from RE
Operand mux
4 32 bit MAC units
128 bit scalar/vector
ALU
control from RE

8.17 TP – Texture Pipe

8.17.1 *Description*

8.17.2 *Major interfaces*

8.17.3 *Block diagram*

8.18 RB – Render Backend

8.18.1 *Description*

8.18.2 *Major interfaces*

8.18.3 *Block diagram*

8.19 MC – Memory Controller

8.19.1 *Description*

8.19.2 *Major interfaces*

8.19.3 *Block diagram*

9. Common Foundations

9.1 Logic Design

9.1.1 *Data formats*

As much as possible, data should be stored and processed identically to x86 (or sparc) conventions. This will, for example, allow the emulator to use normal 32 bit floats and the processors native multiply, add and other operations. This will have a significant effect on the achieved simulation performance compared to being “almost” identical which requires the emulator take several operations to match the hardware bit-exact.

9.1.2 *Register Bus*

Issues:

32 vs. 64 bit
Do rendering state updates happen on this bus or over a dedicated path from memory?
Flow control

9.1.3 *Block Communication protocol*

We want to specify a limited number (one Is probably not possible) number of different ways that blocks are interconnected to simplify verification and emulation.

9.2 Software

Author: Andrew Gruber, Andi Skende

Issue To:	**Copy No:**

Shader Processor

Rev 1.2

Overview: This document describes the overall architecture of the Shaders, interfaces, partitioning into functional blocks as well as the timing of the shader pipeline. It's intended for use by hardware designers.

AUTOMATICALLY UPDATED FIELDS:
Document Location : //ma_andi_mobile/…./doc_lib/parts/sp
Current Intranet Search Title: Shader Processor

APPROVALS

Name/Dept	Signature/Date

Remarks

THIS DOCUMENT CONTAINS CONFIDENTIAL INFORMATION THAT COULD BE SUBSTANTIALLY DETRIMENTAL TO THE INTEREST OF ATI TECHNOLOGIES INC. THROUGH UNAUTHORIZED USE OR DISCLOSURE.

Table Of Contents

STATE 6
1.1 Shader State 6
1.1.1 GPRs (General Purpose Registers) 6
1.1.2 Constant Registers 6
1.1.3 Previous Instruction Result 7
1.2 Initial state 7
1.2.1 Vertex Shader 7
1.2.2 Pixel Shader 7
2. PROGRAM FORMAT 7
3. ALU 7
3.1 ALU structure 7
3.2 ALU instruction format 7
3.2.1 ALU Instruction Word Interpretation 9
3.2.1.1 Relative vs. Absolute Constants 9
3.2.1.2 Argument Selection and Pointers 9
3.2.1.3 Input and Output Modifiers 9
3.2.1.4 Export and Predicate related decoding 10
3.2.1.5 Export Types and Addresses 11
3.3 ALU Opcodes 11
3.4 DX9.0 Shader Instructions, related exceptions and corner cases 14
3.5 Macro opcodes 14
4. SHADER BLOCK DIAGRAMS 15
4.1 Shader as an SIMD architecture 15
4.2 Top-Level Diagram of a Shader Pipeline 15
5. INTERFACES 16
5.1 External Interfaces 16
5.1.1 Naming Convention 16
5.1.2 Shader Engine to Texture Fetch Unit Bus 17
5.1.3 Sequencer to Shader Pipe(s): Texture stall 17
5.1.4 ScanConverter to Shader Pipe: IJ bus 17
5.1.5 Sequencer to Shader Pipe(s) - broadcast: Interpolator bus 18
5.1.6 Sequencer to Shader Pipe(s)-broadcast: Parameter Cache Read control bus 18
5.1.7 Sequencer to Shader Pipe: GPR, Parameter Cache control and auto counter 18
5.1.8 Shader Pipe to Shader Export (SX): Parameter data out of Parameter Cache 19
5.1.9 Shader Export (SX) to Interpolators: Parameter Cache Return bus 19
5.1.10 Shader Pipe to Shader Export (SX): Pixel/Vertex write to SX 19
5.1.11 Sequencer to SPx: Instruction Interface 20
5.1.12 Shader Pipe to Sequencer: Constant address load 22

5.1.13 Sequencer to SPx: constant broadcast ... 22

6. PARAMETER INTERPOLATION ... 23

7. SHADER LIMITATIONS ... 24

8. HARDWARE IMPLEMENTATION SPECIFICS ... 25

8.1 General Information on the Shader Floating Point arithmetic ... 25

8.2 Interpolators and IJ/XY Buffers ... 25

8.2.1 Interpolators ... 26

8.2.1.1 Interpolation Units ... 26

8.2.1.2 Parameter Selection Unit ... 27

8.2.1.3 Parameter Difference & Cylindrical Wrap Engine ... 27

8.2.2 GPR Write Path ... 28

8.3 Vector Unit ... 29

8.3.1 Vector Unit Pipeline ... 29

8.3.2 Argument Selection and Routing ... 30

8.3.3 Parameter Data Path ... 31

8.4 Scalar Unit ... 33

8.4.1 Scalar Engine Pipeline ... 33

8.4.1.1.1.1.1 High Precision Pipeline Exp$_2$ Preprocessing ... 34

8.4.1.1.1.1.2 High Precision Pipeline Mantissa Calculation ... 34

8.4.1.1.1.1.3 High Precision Pipeline Log$_2$ Post Processing ... 38

8.4.1.1.1.1.4 High Precision Pipeline Exponents ... 38

8.4.1.1.1.1.5 High Precision Special Outputs ... 39

8.4.1.1.1.1.6 Determination of High Precision Coefficients ... 39

9. OPEN ISSUES ... 42

Revision Changes:

Rev 0.0 (Steve Morein) Date: Alpril, 2001 Initial revision.	Document started
Rev 0.1 (Andi Skende) Date: May 09, 2001	Updated, added the instruction formant, initial block diagrams and preliminary interface description
Rev 0.2 (Andi Skende) Date: May 21, 2001	A more detailed description of the SP<->TEX, RE/Sequencer <->SP interfaces.
Rev 0.3 (Andi Skende) Date: June 19, 2001	Added the paragraph related to shader functional limitations that the compiler needs to be aware of. A new updated and compressed version of ALU instruction format.
Rev 0.4 (Andi Skende) Date: June 20, 2001	Updated the Introduction of this document. A new Pipeline Timing Diagram was inserted.
Rev 0.5 (Andi Skende) Date: July 31, 2001	Merged in the Shader Hardware Spec. A more detailed description of the interfaces with the other blocks was added. Updated some of the diagrams to a more correct representation of the datapaths.
Rev 0.6 (Andi Skende) Date: August 17,2001	A more detailed description/definition of Shader interfaces with the other blocks. A more detailed description of the instruction supported by Shader Processor and it's relation to instruction set exposed at API level.
Rev 0.7 (Andi Skende) Date: November 8, 2001	Updated the Alu instruction word definition and the list of the alu instruction opcodes supported by the shader pipe ALU unit.
Rev 0.8 (Andi Skende) Date: November 27, 2001	Updated the definition of the External Interfaces
Rev 0.9 (Andi Skende) Date: December 10, 2001	Updated the definition and naming of some of the external interfaces, rearranged the ALU instruction word definition such that the fields are dword aligned. The instruction opcode definition was updated and expanded.
Rev 1.0 (Andi Skende) Date: January 15, 2002	Updated most of the diagrams. Updated the External Interface definitions. Added a description of the Parameter Interpolation Units. Added a diagram desciption of the GPR write data paths.
Rev 1.1 (Andi Skende) Date: January 21, 2002	Updated some of the external interface definitions. Specified the expected behavior of hardware implementation of some shader opcode with some corner case values as input arguments. The MS Reference Rasterizer shader was used as guideline.
Rev 1.2 (Andi Skende) Date: January 22, 2002	Updated some of the external interface definitions.

Introduction

Shader Pipe (SP) serves as the central Arithmetic and Logic Unit (ALU) for the R400 Graphics Processor. There are four identical Shader pipelines in the R400 architecture. Differently from previous ATI architectures, the R400 Shader Pipe truly represents an Unified Shader Architecture. In R400, both vertex and pixel shading operations are implemented through the shader units. The R400 Shader Pipe represents an SIMD architecture. All the shader units of each and every pipe execute the same ALU instruction on different sets of vertex parameters/pixel values. The building blocks of the R400 shader units execute operations on single precision IEEE floating-point values.

State

1.1 Shader State

1.1.1 *GPRs (General Purpose Registers)*

The general-purpose registers are 128 bits wide, composed of four 32-bit values. Depending on the operation these values are interpreted at RGBA, or XYZW, or STQW, or UVQW, or YUVA, or.. to simplify matters the only two aliases used here are XYZW and RGBA.

To hide the latency of memory accesses the shader pipe will switch between different vectors. This is the same as the idea of “microthreading” that some advanced CPU’s are investigating. The large register file is split between the vectors executing in the shader pipe. The management of the shader register file is automatic, and not visible to a program executing on a vector, except that a program is required to declare the number of GPRs it needs to execute. The hardware will not start a vector until the required number of registers is available. There is a direct tradeoff between the number of registers each program/vector needs and the number of vectors than can be simultaneously resident. If there are too few vectors resident, then the latency of memory accesses can no longer be hidden and performance suffers.
There are a total of 128 registers. It is possible for a single program/vector to request all 128 registers. This will make it impossible to hide memory latency, but the program will still execute and generate the correct result.
Most pixel programs are expected to have less than eight registers, vertex programs are expected to have less than sixteen registers.
The number of registers a program needs is the maximum number of registers it needs at any instruction. If a program needs only 3 general purpose registers nearly all of the time, except for a short period when it needs 8, it still needs to allocate eight. A significant performance optimization is for the compiler to reorder the instructions to minimize the number of needed registers.

127	95	63	31 0	GPR
A/W	B/Z	G/Y	R/X	R0
				R1
				R127

Notation: R0.A refers to the bits 96 to 127 of register one. So does R0.W

1.1.2 *Constant Registers*

There are also (192?) constant registers:

127	95	63	31 0	Const
A/W	B/Z	G/Y	R/X	C0
				C1
				C191

These are ONLY available to vertex and pixel shader program in the primary commands stream. They should not be used for real time stream pixel shaders, or 2D shaders. Constant Registers are physically part of the Sequencer unit. As it become clear by reading the rest of this document, the content of the constant registers can be made available to the ALU units of the shader pipes in the form of one of the possible alu operation arguments. ALU instruction word provides for that.

The constant registers are shared between vertex shaders and pixel shaders, it is the drivers job to allocate one section to pixel shaders and another to vertex shaders to match the D3D programming model, other API’s may allow more freedom.
To be able to support multiple textures easily, and to save hardware area, the texture state registers are stored in constant registers. A pair of constant registers hold 256 bits of texture state. Rather than have four or six sets of texture registers as we do in the R100,R200, and R300 by storing them in the constant memory we can save area by reusing the logic already needed to update the constant registers in order. Since any single texture instruction will only fetch from one texture we do not need the simultaneous access we would get with implementing this as “normal” registers. The driver will probably decide to allocate a fixed number of the constant registers as texture registers.

1.1.3 *Previous Instruction Result*

Within an ALU clause the result of the previous operation is explicitly available, without requiring a register read.
(due to an exposed pipeline delay, the result of the previous operation can not be read from the register file without a one-instruction delay slot). There are two distinct previous instructions, one scalar and one vector.
This register is not preserved between the end of one alu clause and the beginning of another.
It can be used to avoid using another GPR if the result is not needed. Also, the output modifiers, which do effect the result of an instruction written into GPRs, do not effect the Previous Result content.

127 95 63 31 0

1.2 Initial state

1.2.1 *Vertex Shader*

A vertex shader initially has the x value of R0 set to the vertex index. No other registers are filled. The vertex shader must use the index to fetch the vertex data from the vertex array(s), The pointers to the vertex arrays should be placed in constant registers by the driver.

1.2.2 *Pixel Shader*

The pixel shader has the interpolated values generated from the values exported by the vertex shader.
If the vertex shader did expxy, and the appropriate control bit in the rasterizer is set, then the register 0 contains the x,y,z,w of the pixel (screen space). If the pixel shader wants a world space x,y,z,w the vertex shader should output that.

2. Program Format

A pixel or vertex shader program consists of 16 clauses, eight texture clauses and eight alu clauses.
The instructions in a clause will be executed sequentially. If a given instruction is implementing, for example,
T * S + D (T = texture for SRC A, S = Specular for Source B, D = Diffuse for Source C), it’s the Sequencer’s task to resolve the dependencies between the ALU clause and the respective texture clause. In other words, the sequencer will not issue the ALU instruction using texture data as input to the shader pipe, until the texture request has been issued to and serviced by the texture pipe. In general, the Shader is not aware of the origine of the SRC A, SRC B and SRC C data (texture, diffuse, specular, vertex parameters etc). Three address pointers into the register files (one for each operand) are all the shaders need to fetch these operands. In reality, as it will become more evident later in this document, there is no need for the pointer values to be passed to the shader units. This is related to the GPR’s read/write mechanism we have chosen to implement.

3. ALU

3.1 ALU structure

ALU consist of two distinct units: the ‘Vector’ ALU and the ‘Scalar’ ALU. The Vector ALU peforms operations in parallel across a 4-component vector, while the Scalar ALU performs operations on a single component of a vector which is then replicated across all components. A single instruction may ‘co-issue’ both a Vector and a Scalar instruction, subject to the limitation that the vector instruction may only require 1 or 2 arguments. For example, a Vector MUL (Multiply) instruction can be coissued with a Scalar instruction, but a MULADD (Multiply and ADD) may not.
For more details on the overall structure of the Shader ALU, refer to the figures in Section 5 of this document.

3.2 ALU instruction format

There are two opcodes present in the ALU instruction, one for the Vector operation and one for Scalar operation. The idea is that we can allow a 4-component vector operation (if the compiler permits) coissued with a Scalar Operation.

The Scalar unit may use SRC C, depening on whether this source is being used by the vector operation. Please refer to Section 8 of this document on the limitations of a Vector or Scalar instruction issuing.

Field	Bits	Size	Description
SRC A Select	95	1	Select bit for selecting Constant vs Register/Vector/Scalar Feedback 0: Constant 1: Register/Previous Vector/Previous Scalar
SRC B Select	94	1	Select bit for selecting Constant vs Register/Vector/Scalar Feedback 0: Constant 1: Register/Previous Vector/Previous Scalar
SRC C Select	93	1	Select bit for selecting Constant vs Register/Vector/Scalar Feedback 0: Constant 1: Register/Previous Vector/Previous Scalar
Vector Opcode	92:88	5	Opcode for Vector instruction
SRC A Register/Constant Pointer	87:80	8	Location of Source A in the register file If not Constant, Bits [6],[7] denote: 00- (absolute register) 01 - (relative register) 10- (previous vector) 11- (previous scalar)
SRC B Register/Constant Pointer	79:72	8	Refer to SRC A Register/Constant Ptr
SRC C Register/Constant Pointer	71:64	8	Refer to SRC A Register/Constant Ptr
Constant0 Relative/Absolute	63	1	The address pointer into the Constant Register File is relative to some base address register (works in conjunction with Address Register Select)
Constant1 Relative/Absolute	62	1	The address pointer into the Constant Register File is relative to some base address register (works in conjunction with Address Register Select)
Relative Address Register Select	61	1	This bit determines the address register used as base register when Constant indexing is relative. It is used in conjunction with **Constan0 Relative/Absolute** and **Constant1 Relative/Absolute** fields. 0:Loop index relative 1:Address Register relative
Predicate Select	60:59	2	This bits are used in conjunction with bit 7 of **Scalar Destination Pointer** and **Vector Destination Pointer**
SRC A Arg Modifier	58	1	0: No modification 1:negate
SRC B Arg Modifier	57	1	0: No modification 1:negate
SRC C Arg Modifier	56	1	0: No modification 1:negate
SRC A swizzle	55:48	8	2 bits for each component **45:46 alpha channel** 00:leave alpha 01:red 10:blue 11:green **47:48 red channel** 00:leave red 01:green 10:blue 11:alpha **49:50 green channel** 00:leave green 01:blue 10:alpha 11:red **51:52 blue channel** 00:leave blue 01:alpha 10:red 11:green
SRC B swizzle	47:40	8	2 bits for each component (refer to 'SRC A swizzle')
SRC C swizzle	39:32	8	2 bits for each component (refer to 'SRC A swizzle')
Scalar Opcode	31:26	6	Opcode for the Scalar instruction
Scalar Clamp	25	1	0: No clamp 1: Clamp to [0.0, 1.0] range
Vector Clamp	24	1	0: No clamp 1: Clamp to [0.0, 1.0] range

Scalar Write Mask	23:20	4	Defines which out of 32 bits words (four of them) in the result is written back in the register file. There's one bit per channel. 0: leave the current value 1: write
Vector Write Mask	19:16	4	Defines which out of 32 bits words (four of them) in the result is written back in the register file. There's one bit per channel 0: leave the current value 1: write
Scalar result pointer	15:8	8	Specifies the address into the register files for the result of scalar operation Bit[6] determines whether the destination address into GPR's is relative or absolute. 0: absolute 1: relative Bit[7] in conjuction with **Predicate Select** are used to define different scenarios of export and predicate functionality. For more on this, refer to Section 3.2.1.2
Vector result pointer	7:0	8	Specifies the address into the register files for the result of vector operation Bit[6] determines whether the destination address into GPR's is relative or absolute. 0: absolute 1: relative

There's a total of 96 bits per instruction.
SrcA, SrcB and SrcC GPR locations denoted by Src A (B, C) Register/Constant Ptr fields of the ALU instruction word, can be relative as well as absolute addresses. If relative, they are relative to a register (Relative Address Register) present in the Sequencer as a render state. The above applies to Constant values as well.
The bit allocation and assignment for the different fields of the instruction word was done with under the limitations that they should be dword (32–bit) aligned.

3.2.1 *ALU Instruction Word Interpretation*

3.2.1.1 Relative vs. Absolute Constants

The location of the Constant Values in the Constant Regiter File can be absolute or relative to an offset value. When relative, they can be relative to either a loop index or a given register content value. The truth table shows the instruction fields that are used to decode the nature of the constant values.

Constant0 Relative/Absolute	**Constant1 Relative/Absolute**	**Address Register Select**	**Notes**
0	0	0	Constant0 –absolute Constant1-absolute
0	0	1	Constant0 --absolute,9 bits, Constant 1 Absolute
1	0	0	Constant0 –loop index relative Constant1-absolute
0	1	0	Constant0-absolute Constant1 –loop index relative
1	1	0	Constant0–loop index relative Constant0–loop index relative
1	0	1	Constant0-address relative Constant1-absolute
0	1	1	Constant0-absolute Constant1-address relative
1	1	1	Constant0-address relative Constant1-address relative

Note from the table that if both Constants are relative, they are relative to the same value, being that a loop index or address register.

3.2.1.2 Argument Selection and Pointers

There can be a maximum of three sources (operands) required for an ALU operation of a vector type.
The R400 ALU instruction word definition provides location pointers into GPRs or Constant Memory for each of the three sources (**SRC A Register/Constant Pointer, SRC B Register/Constant Pointer**, **SRC B Register/Constant Pointer).**

3.2.1.3 Input and Output Modifiers

The R400 ALU Instruction word definition provides for only two input modifiers for each of the three sources, **Negate** and **Swizzle**.
The R400 ALU Instruction word provides for two output (result) modifiers: **Mask** which only effects the results going into GPRs but not the Previous Results and **Clamp**.

3.2.1.4 Export and Predicate related decoding

The table below describes the encoding of the exports and predicate support in the instruction word.

Exports are allowed from either Scalar or Vector Pipe. Similar to the GPR write-backs, masking of export data is permitted. The mask is present in the ALU instruction word. In cases when exports are coissued from Scalar and Vector pipes, the export address used is the **Vector Result Pointer** present in the instruction word. The Scalar Result Pointer in this case is ignored. The table below describes the “mixed” use of Scalar and Vector Result Masks per component (each MASK field is 4 bits wide, one bit per component/channel) when exports are coissued.

Scalar Mask	Vector Mask	Result of Export
0	0	Don’t write
1	0	Write Scalar Component
0	1	Write Vector Component
1	1	Write 1.0 (one way of generating defaults)

A few other export related definitions:

1) When doing a Scalar export of 'pixels' or 'position', only the 'alpha' component will contain the scalar result. The other 3 components will be expanded to 0.0. When exporting to 'parameters' the scalar result is put into all 4 components.
2) When doing a Scalar export of 'parameters', non-export vector instructions may not be coissued.
3) Exporting of 'Fog' is a special case.
    a) When exporting Fog, color must be exported at the same time. Fog will be exported in the Scalar pipe and Color in the Vector pipe.
    b) Masking is ignored for Fog exports. Instead Color and Fog are mixed into a single ARGBF word and exported to the render back-end.

Predicate Select	Scalar Destination Pointer Bit[7]	Vector Destination Pointer Bit[7]	Notes
0X	1	0	Scalar Export Vector to GPR
0X	1	1	Scalar Export Vector Export
0X	0	0	Scalar to GPR Vector to GPR
0X	0	1	Scalar to GPR Vector Export
10	0	0	Scalar to GPR Vector to GPR Use Predicate register0 1: skip 0: execute
10	0	1	Scalar to GPR Vector to GPR Use Predicate Register0 1: execute 0: skip
10	1	0	Scalar to GPR Vector to GPR Use Predicate Register1 1: skip 0: execute
10	1	1	Scalar to GPR Vector to GPR Use Predicate Register1 1: execute 0: skip
11	0	0	Scalar to GPR Vector to GPR Use Predicate Register2 1: skip 0: execute
11	0	1	Scalar to GPR Vector to GPR Use Predicate Register2 1: execute 0: skip
11	1	0	Scalar to GPR Vector to GPR Use Predicate register3 1: skip 0: execute
11	1	1	Scalar to GPR Vector to GPR Use Predicate register3 1: execute 0: skip

3.2.1.5 Export Types and Addresses

The location where the data should be put in the event of an export is specified by in the destination address field of the ALU instruction word. Following is a list of the possible types of exports and the range of addresses.

Vertex Shading

0:15	- 16 parameter cache
16:31	- Empty (Reserved?)
32:43	- 12 vertex exports to the frame buffer and index
44:47	- Empty
48:59	- 12 debug export (interpret as normal vertex export)
60	- export addressing mode
61	- Empty
62	- sprite size export that goes with position export (point_h,point_w,edgeflag,misc)
63	- position

Pixel Shading

0	- Color for buffer 0 (primary)
1	- Color for buffer 1
2	- Color for buffer 2
3	- Color for buffer 3
4:7	- Empty
8	- Buffer 0 Color/Fog (primary)
9	- Buffer 1 Color/Fog
10	- Buffer 2 Color/Fog
11	- Buffer 3 Color/Fog
12:15	- Empty
16:31	- Empty (Reserved?)
32:43	- 12 exports for multipass pixel shaders.
44:47	- Empty
48:59	- 12 debug exports (interpret as normal pixel export)
60	- export addressing mode
61:62	- Empty
63	- Z for primary buffer (Z exported to 'alpha' component)

3.3 ALU Opcodes

The following table represents the ALU operations/opcodes supported by the Vector unit.

Name	Opcode	Function	Notes
ADD	0x00	Result = A + B	2 operand instruction; possible coissue
MUL	0x01	Result = A *B	2 operand instruction, possible coissue
MAX	0x02	If (A >= B) result = A; else result = B	
MIN	0x03	If (A < B) result = A; else result = B	
SETE	0x04	If (A = B) result = 1.0; else result = 0.0	
SETGT	0x05	If (A > B) result = 1.0; else result = 0.0	
SETGE	0x06	If (A >=B) result = 1.0; else result = 0.0	
SETNE	0x07	If (A != B) result = 1.0; else result = 0.0	
FRACT	0x08	Result = fractional part of A	
TRUNC	0x09	Result = integer part of A	
FLOOR	0x0a	Result = TRUNCT(A) for positive A, (TRUNC(A) –1) for negative A	
MULADD	0x0b	Result = A * B + C	3 operand instruction; no coissue
CNDE	0x0c	If (A == 0.0) result = B ; else result = C	3 operand instruction; no coissue
CNDGE	0x0d	If (A >= 0.0) result = B; else result = C	3 operand instruction; no coissue
CNDGT	0x0e	If (A > 0.0) result = B; else result = C	3 operand instruction; no coissue
DOT4	0x0f	Result = A dot B in 4 components	Result replicated in all channels
DOT3	0x10	Result = A dot B in 3 components	Result replicated in all four channels
CUBE	0x11	Result.alpha = max(A.red, A.green, A.blue);	

		Result.blue = faceID code;	
MAX4	0x13	Result.alpha = max(A.red,A.green,A,blue,A.alpha)	

DOT4ABS	0x14	Result.a = abs(a.alpha*b.alpha) + abs(a.red*b.red) + abs(a.green*b.green) + abs(a.blue*b.green)	The idea is that you woul execute:
			ADD R0, R0, R1, neg ; // R0= R0 – R1
			DOT4ABS R0.a, R0, C0;
			// C0 holds (1.0,1.0,1.0,1.0).
			// R0.a = abs(R0.a) + abs(R0.r) + abs(R0.g) + abs(R0.b). It saves from doing an abs( ) between ADD and DOT4 in order to implement a sum of absolute differences for 4 values.

DOT4SUB	0x15	Result.a = A.alpha * B.alpha – (A.red * B.red + A.blue*B.blue + A.green * B.green)	
PRED_SETE	0x16	If(A ==B) result =1.0; else result = 0.0	The result is written to predicate register 'n', not the register file. Note that A==B compare is a per channel compare. As such, if any of the compares yields a result of 1.0 then the predicate bit is set.
PRED_SETGT	0x17	If(A > B) result = 1.0; else result = 0.0	Same as above
PRED_SETGE	0x18	If(A >= B) result =1.0; else result = 0.0	Same as above
PRED_SETNE	0x19	If(A != B) result = 1.0; else result = 0.0	Same as above
MASKE	0x1a	If(A ==B) result =1.0 ;else result= 0.0	The result is used to determine the visibility of the pixel, and is not to be written back to the register file. A 1.0 indicates that the pixel is made not visible. Once a pixel is masked, subsequent instructions can't 'unmask' it. Note that A==B compare is a per channel compare. As such, if any of the compares yields a result of 1.0 then the predicate bit is set.
MASKGT	0x1b	If(A > B) result =1.0 ;else result= 0.0	Same as above
MASKGE	0x1c	If(A >=B) result =1.0 ;else result= 0.0	Same as above
MASKNE	0x1d	If(A != B) result = 1.0; else result = 0.0	Same as above

MOV instruction

MOV instruction can be implemented via a MIN or MAX instruction using argument selection bits in the ALU instruction appropriately.

Predicate instructions

There are three predicate instructions. In the case of a predicate instruction, the result is written into a predicate register, not into a register file. There are four predicate registers, so the destination must be specified in the range 0-3. For more speficics on this, please refer to the Sequencer's architecture speficication document.

The following table represents the ALU operations and their respective opcodes supported by the Scalar Unit.

Name	Opcode	Function	Notes
ADD	0x00	Result = SRC.alpha + SRC.red	
ADD_PREV	0x01	Result = SRC.alpha + PreviousScalar	
MUL	0x02	Result = SRC.alpha * SRC.red	
MUL_PREV	0x03	Result = SRC.alpha * PreviousScalar	

MAX	0x04	If (SRC.alpha >= SRC.red)
		Result = SRC.alpha;
		Else
		Result = SRC.red	

MIN	0x05	If (SRC.alpha < SRC.red)
		Result = SRC.alpha;
		Else
		Result = SRC.red;	

SETE	0x06	If (SRC.alpha = SRC.red) Result = 1.0;	

		Else Result = 0.0;	

SETGT	0x07	If (SRC.alpha > SRC.red)
		Result = 1.0;
		Else
		Result = 0.0;	

SETGE	0x08	If (SRC.alpha >= SRC.red)
		Result = 1.0;
		Else
		Result = 0.0;	

SETNE	0x09	If (SRC.alpha ! = SRC.red)
		Result = 1.0;
		Else
		Result = 0.0;	

FRACT	0x0a	Result = fractional part of SRC.alpha	
TRUNC	0x0b	Result = integer part of SRC.alpha	
FLOOR	0x0c	Result = TRUNCT(SRC.alpha) for positive SRC.alpha, (TRUNC(SRC.alpha) –1) for negative SRC.alpha	
EXP	0x0d	Result = 2^ (SRC.alpha) ??	Base 2 exponent function
LOG	0x0e	Result = log(SRC.alpha)	Base 2 log function
RECIP	0x0f	Result = 1/SRC.alpha;	
RECIPSQRT	0x10	Result = 1/sqrt(SRC.alpha)	
MOVA	0x1e	FLOOR(SRC.alpha)	This (9 bit) value is then exported to the addressing registers in the sequencer
SUB	0x12	Result = Src.alpha – Src.red	This instruction is needed since NEG argument modifier, part of ALU instruction, applies to all channels of the source.
SUB_PREV	0x13	Result = Src.alpha – PreviousScalar	
PRED_SETE	0x14	If(Source.alpha ==Source.red) result =1.0; else result = 0.0	The result is written to predicate register ‘n’, not the register file.
PRED_SETGT	0x15	If(Source.alpha > Source.red) result = 1.0; else result = 0.0	Same as above
PRED_SETGE	0x16	If(Source.alpha >= Source.red) result =1.0; else result = 0.0	Same as above
PRED_SETNE	0x17		
PRED_SETZ	0x18	If(Source.alpha == 0.0) result = 1.0 ; else result = 0.0	
PRED_SETONE	0x19	If(Source.alpha == 1.0) result = 1.0 ; else result = 0.0	

MASKE	0x1a	If(Source.alpha ==Source.red)
		result =1.0 ;
		else
		result= 0.0
			The result is used to determine the visibility of the pixel, and is not to be written back to the register file. A 1.0 indicates that the pixel is made not visible. Once a pixel is masked, subsequent instructions can not ‘unmask’ it.

MASKGT	0x1b	If(Source.alpha >Source.red) result =1.0 ;else result= 0.0	Same as above
MASKGE	0x1c	If(Source.alpha >=Source.red) result =1.0 ;else result= 0.0	Same as above
MASKNE	0x1d	If(Source.alpha !=Source.red) result =1.0 ;else result= 0.0	Same as above
MASKZ	0x1e	If(Source.alpha == 0.0) result = 1.0 ; else result = 0.0	Same as above
MASKONE	0x1f	If(Source.alpha == 1.0) result = 1.0 ; else result = 0.0	Same as above
SQRT	0x20	Result = EXP(0.5 * LOG(Source.alpha))	Useful for normal compression

Note: a **MOV** instruction can be implemented via a MIN or MAX instruction using argument selection bits in the ALU instruction appropriately.

3.4 DX9.0 Shader Instructions, related exceptions and corner cases

The following list of shader opcodes and their outputs for corner case values are based on the reference shader code out of the DX9.0 Reference Rasterizer. R400 shader implementation in hardware will have the same behavior.

1. RCP opcode (Reciprocal)
RCP(Src.w), where Src.w = 0.0, is defined as the maximum value that can be represented by a single precision IEEE floating point number. The hex representation of this number would be 0x7F7FFFFF.
2. RECIPSQRT opcode (Reciprocal of Square Root)
RECIPSQRT(Src.w), where Src.w = 0.0, is defined as the maximum value that can be represented by a single precision IEEE floating point number. The hex representation of this number would be 0x7F7FFFFF.
3. LOG opcode (base 2 LOG)
LOG(Src.w) (base 2 LOG), where Src.w = 0.0, is defined as the smallest values that can be represented by a single precision IEEE floating point number. The hex representation of this number would be 0xFF7FFFFF. For the rest of the values, LOG(Src.w) is defined as LOG(abs(Src.w)). In other words, the sign of the Src values is ignored.

3.5 Macro opcodes

Instructions that API (DirectX, etc) defines as implementable via more “basic” instructions are know as MACROs. DirectX 9.0 defines a few MACRO opcodes. For example:

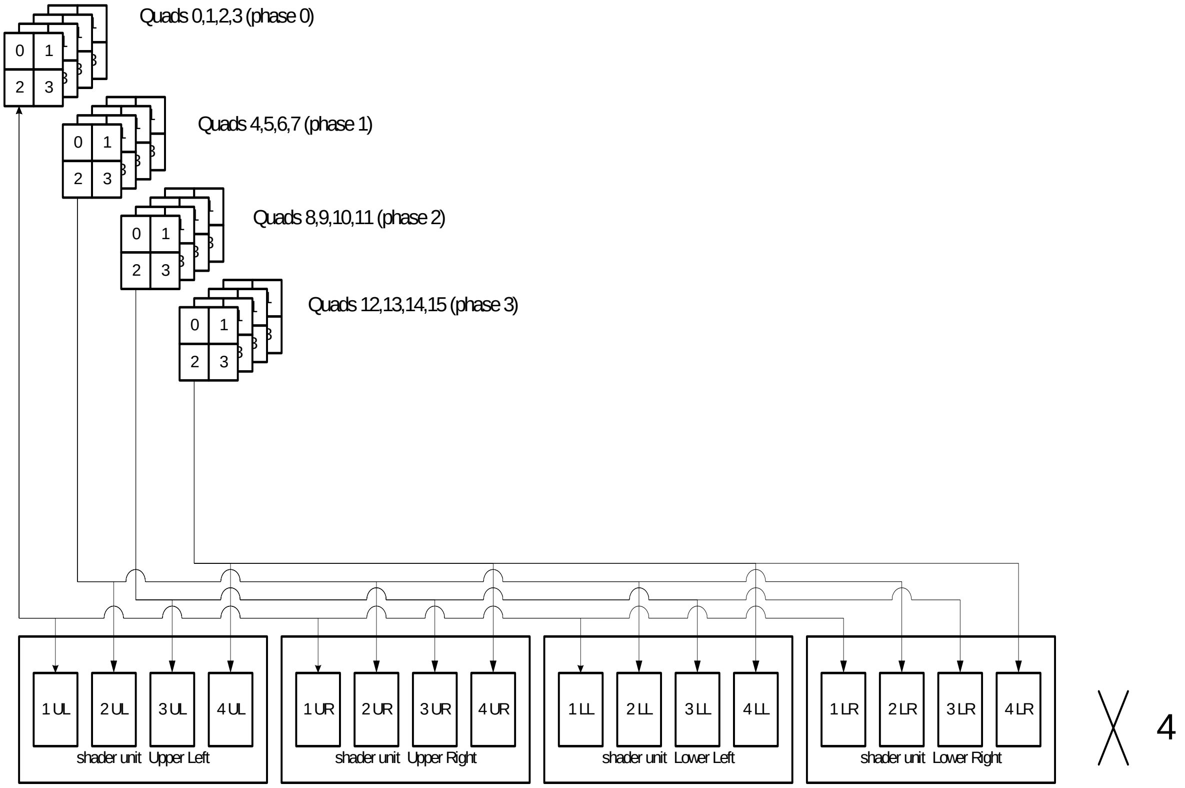
POW - vector x power y
Instruction: POW DST, SRC1, SRC2
Description: Computes x power y.
Input power must be a scalar. Scalar result is replicated to all 4 output channels.
This is a macro instruction, which takes 3 instruction slots.
pow(x,y) could be expanded as exp(y * log(x)).
DST should be a temporary register
Macro expansion:
log DST, SRC1
mul DST, DST, SRC1
exp DST, DST

4. Shader Block Diagrams

4.1 Shader as an SIMD architecture

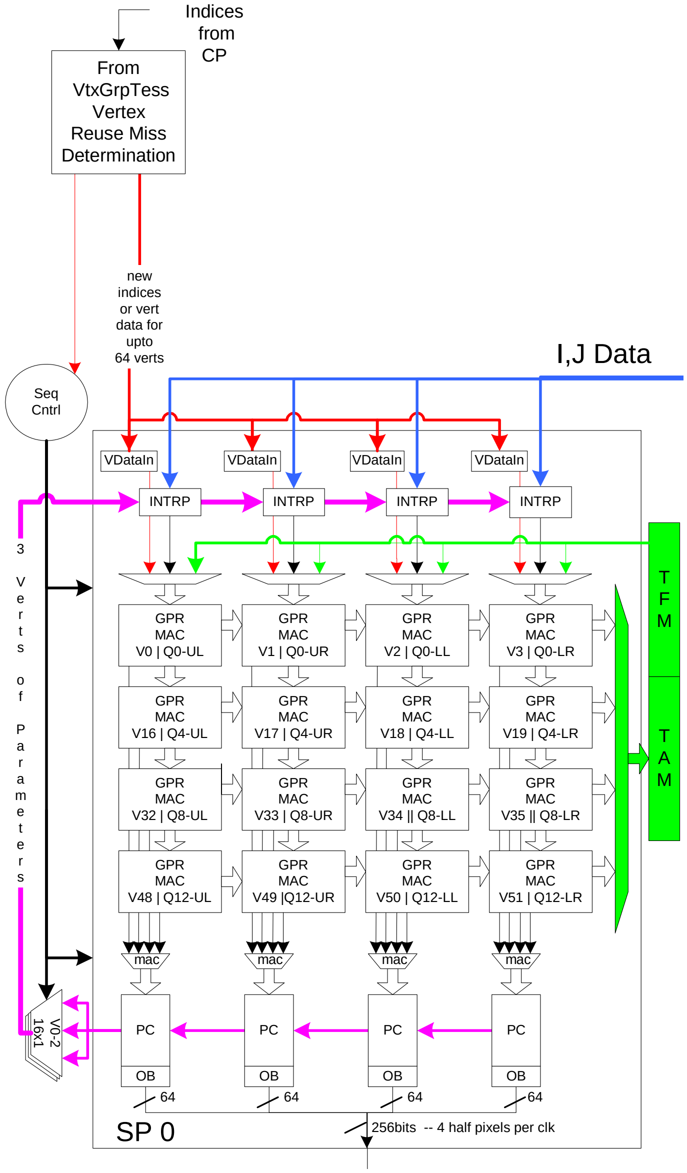
As shown in the diagram below, four identical processing units comprise a shader unit. There are four shader units in one shader pipeline. R400 has four shader pipelines. The full R400 shader pipe represents an example of a SIMD (**S**ingle **I**nstruction **M**ultiple **D**ata streams) architecture: the four shader pipelines, each with 4 identical shader units, executing the same pair of Vector/Scalar Instruction on different data streams, in this case different pixel positions within the same quad, over four different quads of pixels in parallel. In figure, the shader units are named as Upper Left, Upper Right, Lower Left and Lower Right based on the relative position within the quad of the pixels they process.

Within one shader pipe, only 4 processing units, one from each shader unit, are in the same execution sequence (phase) at a given time. So, through the whole chip we have 4 sets of 16 processing units being at different execution phases from one set to the other, but in the same execution phase within the same set.



4.2 Top-Level Diagram of a Shader Pipeline

The diagram below represents a high level description of a shader pipe. The major data streams coming into or going out of it are clearly shown.



5. Interfaces

5.1 External Interfaces

5.1.1 *Naming Convention*

TP -> stands for Texture Pipe.
SP-> stands for Shader Pipe
SQ->stands for Sequencer Unit
SX->stands for Shader Export

SC->stands for Scan Converter
When an X is used as a postfix in the Direction colomn of the interface definition tables, it means that the bus is a broadcast bus to all units with the same name. For example, if the direction of the bus is defined as SQ->SPx, that means that Sequencer is broadcasting the same information to all Shader Pipe instances.

5.1.2 *Shader Engine to Texture Fetch Unit Bus*

Four quad's worth of addresses is transferred to Fetch Unit every clock. These are sourced from a different pixel within each of the sub-engines repeating every 4 clocks. The register file index to read must precede the data by 2 clocks. The Read address associated with Quad 0 must be sent 1 clock after the Instruction Start signal is sent, so that data is read 3 clocks after the Instruction Start.

Four Quad's worth of Fetch Data may be written to the Register file every clock. These are directed to a different pixel of the sub-engines repeating every 4 clocks. The register file index to write must accompany the data. Data and Index associated with the Quad 0 must be sent 3 clocks after the Instruction Start signal is sent.

Name	Direction	Bits	Description
SP0_TP0_fetch_addr	SP0->TP0	384	3 Fetch Addresses read from the Register file
TP0_SP0_data	TP0→SP0	512	4 texture results
SP1_TP1_fetch_addr	SP1->TP1	384	3 Fetch Addresses read from the Register file
TP1_SP1_data	TP1→SP1	512	4 texture results
SP2_TP2_fetch_addr	SP2->TP2	384	3 Fetch Addresses read from the Register file
TP2_SP2_data	TP2→SP2	512	4 texture results
SP3_TP3_fetch_addr	SP3->TP3	384	3 Fetch Addresses read from the Register file
TP3_SP3_data	TP3→SP3	512	4 texture results
TPx_SPx_gpr_dst	TPx→SPx	7	Write address into the gprs
TPx_SPx_gpr_cmask	TPx→SPx	4	Channel mask. Supports the ability to mask any of the 32 bit channel of the fetch return data

5.1.3 *Sequencer to Shader Pipe(s): Texture stall*

Texture pipe signals the Sequencer that its input buffer is full. The Sequencer asserts SQ_SPx_fetch_stall so that Shader Pipe does not send new requests to the Texture Pipe.

Name	Direction	Bits	Description
SQ_SPx_fetch_stall	SQ→SPx	1	Do not send more texture requests if asserted

5.1.4 *ScanConverter to Shader Pipe: IJ bus*

This is a bus that sends the IJ information to the IJ fifos on the top of each shader pipe. At the same time the control information goes to the sequencer. There are 4 of these buses over the whole chip (SP0 thru 3)

Name	Direction	Bits	Description
SC_SP0_data	SC→SP0	64	IJ information sent over 2 clocks (or XY info sent over 1 clock in the lower 24 LSBs of the interface)
SC_SP0_q_wr_mask	SC→SP0	1	Write Mask
SC_SP0_dest	SC→SP0	1	Controls the write destination (XY buffer, IJ buffer)
SC_SP1_data	SC→SP1	64	IJ information sent over 2 clocks (or XY info sent over 1 clock in the lower 24 LSBs of the interface)
SC_SP1_q_wr_mask	SC→SP1	1	Write Mask
SC_SP1_dest	SC→SP1	1	Controls the write destination (XY buffer, IJ buffer)
SC_SP2_data	SC→SP2	64	IJ information sent over 2 clocks (or XY info sent over 1 clock in the lower 24 LSBs of the interface)
SC_SP2_q_wr_mask	SC→SP2	1	Write Mask
SC_SP2_dest	SC→SP2	1	Controls the write destination (XY buffer, IJ buffer)
SC_SP3_data	SC→SP3	64	IJ information sent over 2 clocks (or XY info sent over 1 clock in the lower 24 LSBs of the interface)
SC_SP3_q_wr_mask	SC→SP3	1	Write Mask
SC_SP3_dest	SC→SP3	1	Controls the write destination (XY buffer, IJ buffer)

SC_SQ_RTS	SC→SQ	1	SC ready to send data

5.1.5 *Sequencer to Shader Pipe(s) - broadcast: Interpolator bus*

This bus interface defines all the signals needed to perform the interpolation of the primitive parameters coming from the Parameter Caches via the SX blocks.

Name	Direction	Bits	Description
SQ_SPx_interp_prim_type	SQ→SPx	3	Type of the primitive 000 : Normal 011 : Real Time 100 : Line AA 101 : Point AA 110 : Sprite
SQ_SPx_interp_flat_vtx	SQ→SPx	2	Provoking vertex for flat shading
SQ_SPx_interp_flat_gouraud	SQ→SPx	1	Flat or gouraud shaded interpolation
SQ_SPx_interp_cyl_wrap	SQ→SPx	4	Which channel of the parameter being interpolated needs to be wrapped
SQ_SPx_interp_ijline	SQ→SPx	2	Line in the IJ/XY buffer to use to interpolate
SQ_SPx_interp_buff_swap	SQ→SPx	1	Swap the IJ/XY buffers at the end of the interpolation
SQ_SPx_interp_gen_I0	SQ→SPx	1	Generate I0 or not. This tells the interpolators not to use the parameter cache but rather overwrite the data with interpolated 1 and 0. Overwrite if gen_I0 is high.

5.1.6 *Sequencer to Shader Pipe(s)-broadcast: Parameter Cache Read control bus*

This interface provides three different pointers specifying the location of the parameter values in the Parameter Caches. Depending on the way the vertices get mapped into primitives, it might happen that the parameter values come from different relative offsets in the parameter caches from one parameter cache to the other across a shader pipe. This is the reason why three different read pointers are specified. This is not true for the write path.
This is a broadcast interface.

Name	Direction	Bits	Description
SQ_SPx_ptr0	SQ→SPx	7	Parameter Pointer into PC
SQ_SPx_ptr1	SQ→SPx	7	Parameter Pointer into PC
SQ_SPx_ptr2	SQ→SPx	7	Parameter Pointer into Parameter Cache
SQ_SPx_pc0_addr_sel	SQ→SPx	2	Selection one of the pointers for parameter cache 0
SQ_SPx_pc1_addr_sel	SQ→SPx	2	Selection one of the pointers for parameter cache 1
SQ_SPx_pc2_addr_sel	SQ→SPx	2	Selection one of the pointers for parameter cache 2
SQ_SPx_pc3_addr_sel	SQ→SPx	2	Selection one of the pointers for parameter cache 3
SQ_SP0_read_ena	SQ→SP0	4	Read enables for the 4 memories in the SP0
SQ_SP1_read_ena	SQ→SP1	4	Read enables for the 4 memories in the SP1
SQ_SP2_read_ena	SQ→SP2	4	Read enables for the 4 memories in the SP2
SQ_SP3_read_ena	SQ→SP3	4	Read enables for the 4 memories in the SP3

5.1.7 *Sequencer to Shader Pipe: GPR, Parameter Cache control and auto counter*

This interface defines the control mechanism for the GPR read/write paths as well as the Parameter Cache write path.

Name	Direction	Bits	Description
SQ_SPx_wr_addr	SQ→SPx	7	Write address (the same bus used for writing into GPRs or Parameter Cache)
SQ_SPx_gpr_rd_addr	SQ→SPx	7	Read address

SQ_SPx_gpr_re_addr	SQ→SPx	1	Read Enable
SQ_SPx_gpr_we_addr	SQ→SPx	1	Write Enable for the GPRs
SQ_SPx_gpr_phase_mux	SQ→SPx	2	The phase mux (arbitrates between inputs, ALU source reads and writes)
SQ_SPx_channel_mask	SQ→SPx	4	The channel mask
SQ_SP0_pixel_mask	SQ→SP0	4	The pixel mask
SQ_SP1_pixel_mask	SQ→SP1	4	The pixel mask
SQ_SP2_pixel_mask	SQ→SP2	4	The pixel mask
SQ_SP3_pixel_mask	SQ→SP3	4	The pixel mask
SQ_SPx_pc_we_addr	SQ→SPx	1	Write Enable for the parameter caches
SQ_SPx_gpr_input_mux	SQ→SPx	2	When the phase mux selects the inputs this tells from which source to read from: Interpolated data, VTX0, VTX1, autogen counter.
SQ_SPx_index_count	SQ→SPx	12?	Index count, common for all shader pipes

5.1.8 *Shader Pipe to Shader Export (SX): Parameter data out of Parameter Cache*

There is 512-bit of data (4 x128) coming out of each shader pipe for each read out of the parameter caches. These data gets routed into the interpolation units by the SX blocks.

Name	**Direction**	**Bits**	**Description**
SP0_SX0_data0	SP0→SX0	128	Parameter data 0
SP0_SX0_data1	SP0→SX0	128	Parameter data 1
SP0_SX0_data2	SP0→SX0	128	Parameter data 2
SP0_SX0_data3	SP0→SX0	128	Parameter data 3
SP1_SX1_data0	SP1→SX1	128	Parameter data 0
SP1_SX1_data1	SP1→SX1	128	Parameter data 1
SP1_SX1_data2	SP1→SX1	128	Parameter data 2
SP1_SX1_data3	SP1→SX1	128	Parameter data 3
SP2_SX0_data0	SP2→SX0	128	Parameter data 0
SP2_SX0_data1	SP2→SX0	128	Parameter data 1
SP2_SX0_data2	SP2→SX0	128	Parameter data 2
SP2_SX0_data3	SP2→SX0	128	Parameter data 3
SP3_SX1_data0	SP3→SX1	128	Parameter data 0
SP3_SX1_data1	SP3→SX1	128	Parameter data 1
SP3_SX1_data2	SP3→SX1	128	Parameter data 2
SP3_SX1_data3	SP3→SX1	128	Parameter data 3

5.1.9 *Shader Export (SX) to Interpolators: Parameter Cache Return bus*

This bus represents the values of a given parameter at the three vertices of the primitive.
Note: The nature of this bus might change in the future depending on where the Parameter Difference engine physically resides (see Section of this document titled “Open Issues”).

Name	**Direction**	**Bits**	**Description**
SXx_SPx_vtx_data_0	SXx→SPx	128	Vertex data to interpolate
SXx_SPx_vtx_data_1	SXx→SPx	128	Vertex data to interpolate
SXx_SPx_vtx_data_2	SXx→SPx	128	Vertex data to interpolate

5.1.10 *Shader Pipe to Shader Export (SX): Pixel/Vertex write to SX*

Name	**Direction**	**Bits**	**Description**

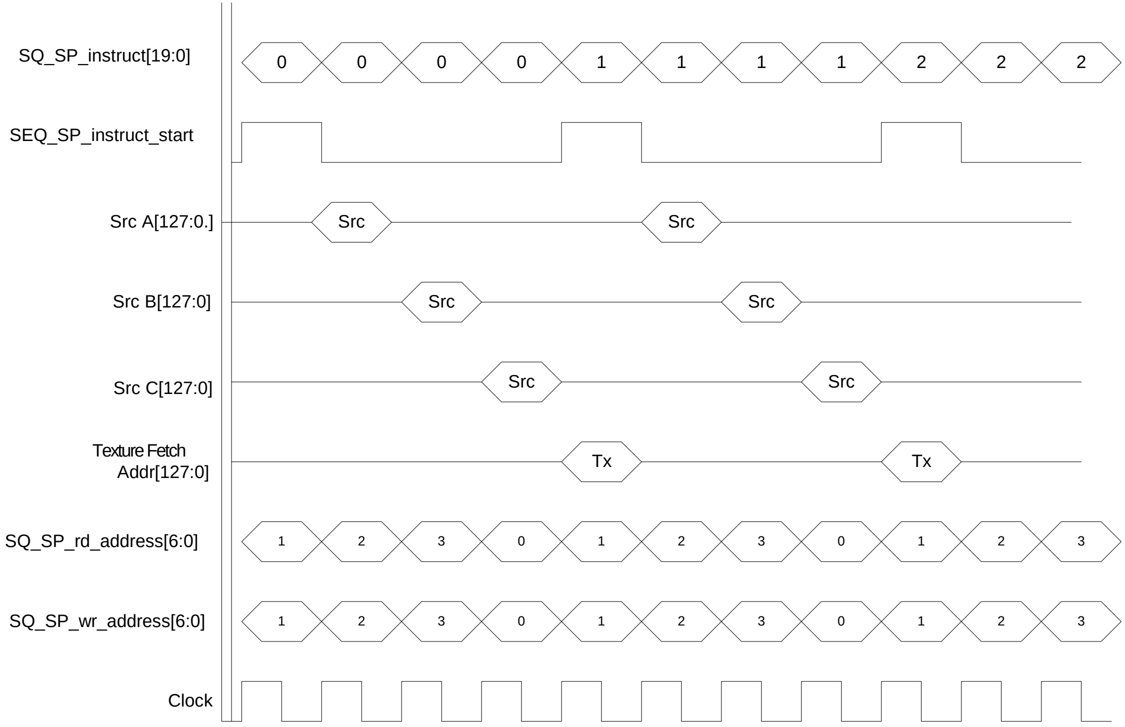
SP0_SX0_export_data	SP0→SX0	256	4 pairs of 32 bits channel values
SP0_SX0_export_dst	SP0→SX0	4	Specifies one of the of up to 12 export destinations
SP1_SX1_export_data	SP1→SX1	256	4 pairs of 32 bits channel values
SP1_SX1_export_dst	SP1→SX1	4	Specifies one of the of up to 12 export destinations
SP2_SX0_export_data	SP2→SX0	256	4 pairs of 32 bits channel values
SP2_SX0_export_dst	SP2→SX0	4	Specifies one of the of up to 12 export destinations
SP3_SX1_export_data	SP3→SX1	256	4 pairs of 32 bits channel values
SP3_SX1_export_dst	SP3→SX1	4	Specifies one of the of up to 12 export destinations
SPx_SXx_ export _count	SP0→SX0	3	Each set of four pixels or vectors is exported over eight clocks. This field specifies where the SP is in that sequence.
SPx_SXx_ export _last	SP0→SX0	1	Asserted on the first shader count of the last export of the clause
SP0_SX0_ export _pvalid	SP0→SX0	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SP0_SX0_ export _wvalid	SP0→SX0	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors
SP1_SX1_ export _pvalid	SP1→SX1	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SP1_SX1_ export_wvalid	SP1→SX1	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors
SP2_SX0_ export _pvalid	SP2→SX0	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SP2_SX0_ export _wvalid	SP2→SX0	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors
SP3_SX1_ export _pvalid	SP3→SX1	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SP3_SX1_ export _wvalid	SP3→SX1	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors

5.1.11 *Sequencer to SPx: Instruction Interface*

This is the bus that sends the instruction and constant data to all four Shader pipe instances. Because a new instruction is needed only every 4 clocks, the width of "SQ_SPx_instruct" sub-bus is divided by 4 and both constants and instruction are sent over those 4 clocks. **SRC A (B or C) Select** of SQ_SP_Instruction interface is derived by Sequencer from the **SRC A (B,C)** and **SRC A (B,C) Register/Constant Pointer** of the ALU Instruction word. All the other bit-fields in the SQ_SP_instruct interface bus are explicitly present in the ALU Instruction word. Please refer to Section 3.2 of this document for more details on the R400 ALU instruction word format.

Name	Direction	Bits	Description
SQ_SPx_instruct_start	SQ→SPx	1	Instruction start
SQ_SPx_instruct	SQ→SPx	20	Transferred over 4 cycles 0: SRC A Select 2:0 SRC A Argument Modifier 3:3 SRC A swizzle 11:4 Unused 19:12 --- -- 1: SRC B Select 2:0 SRC B Argument Modifier 3:3 SRC B swizzle 11:4

			Unused 19:12 -- -- 2: SRC C Select 2:0 SRC C Argument Modifier 3:3 SRC C swizzle 11:4 Unused 19:12 -- -- 3: Vector Opcode 4:0 Scalar Opcode 9:5 Vector Clamp 10:10 Scalar Clamp 11:11 Vector Write Mask 15:12 Scalar Write Mask 19:16
SQ_SPx_stall	SQ→SPx	1	Stall signal (ALU executes a Max PV,PV and Max PS,PS instruction)
SQ_SPx_export_count	SQ→SPx	3	Each set of four pixels or vectors is exported over eight clocks. This field specifies where the SP is in that sequence.
SQ_SPx_export_last	SQ→SPx	1	Asserted on the first shader count of the last export of the clause
SQ_SP0_export_pvalid	SQ→SP0	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP0_export_wvalid	SQ→SP0	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors
SQ_SP1_ export_pvalid	SQ→SP1	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP1_ export_wvalid	SQ→SP1	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors
SQ_SP2_ export_pvalid	SQ→SP2	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP2_ export_wvalid	SQ→SP2	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors
SQ_SP3_ export_pvalid	SQ→SP3	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP3_ export_wvalid	SQ→SP3	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors



The above diagram attempts to describe the timing relation between the signals at SQ-SP instruction interface. Each instruction (SQ_SP_instruct [19:0]) is broadcasted over four cycles. SQ_SP_instruct_start is asserted at the first cycle of the instruction broadcast. Cycle 0 of SQ_SP_rd_address is “dedicated” to reading SRC A out of the Register File, Cycle 1 to reading SRC B, Cycle 2 to reading SRC C and Cycle 3 to reading Texture Address for a texture fetch request. Cycle 0 of SQ_SP_wr_address is “dedicated” to writing into Register File of Interpolated data from Interpolation units, Cycle 1 to writing of return data from a previously issued texture fetch and Cycle 3 to Previous Vector result and Cycle 3 to Previous Scalar result.

5.1.12 *Shader Pipe to Sequencer: Constant address load*

Name	Direction	Bits	Description
SP0_SQ_const_addr	SP0→SQ	36	Constant address load to the sequencer
SP0_SQ_valid	SP0→SQ	1	Data valid
SP1_SQ_const_addr	SP1→SQ	36	Constant address load to the sequencer
SP1_SQ_valid	SP1→SQ	1	Data valid
SP2_SQ_const_addr	SP2→SQ	36	Constant address load to the sequencer
SP2_SQ_valid	SP2→SQ	1	Data valid
SP3_SQ_const_addr	SP3→SQ	36	Constant address load to the sequencer
SP3_SQ_valid	SP3→SQ	1	Data valid

5.1.13 *Sequencer to SPx: constant broadcast*

The interface represents the constant values interface coming from the Sequencer unit. Constant values can be selected as operands for in a given shader instruction.

Name	**Direction**	**Bits**	**Description**
SQ_SPx_constant	SQ→SPx	128	Constant broadcast

6. Parameter Interpolation

This section was partially copied from Section 15 "IJ Format" of the "R400 Sequencer Specification" document. There are two key featues in R400 interpolation scheme:

a. Barycentric coordinates are used in the interpolation of all parameters.
b. The interpolation is done at a different precision across a 2x2 colletion of pixels. The parameters of the upper left pixel of the quad are interpolated at full 20x24 mantissa precision. Then the result along with the difference in IJ barycentric coords is used to interpolate the parameters for the remaining pixels of the 2x2.

Assuming P0 is the interpolated parameter at Pixel 0 having the barycentric coordinates I(0), J(0) and so on for P1,P2 and P3. Also assuming that A is the parameter value at V0 (interpolated with I), B is the parameter value at V1 (interpolated with J) and C is the parameter value at V2 (interpolated with (1-I-J).

$$\Delta 01I = I(1) - I(0)$$
$$\Delta 01J = J(1) - J(0)$$
$$\Delta 02I = I(2) - I(0)$$
$$\Delta 02J = J(2) - J(0)$$
$$\Delta 03I = I(3) - I(0)$$
$$\Delta 03J = J(3) - J(0)$$

$$P0 = C + I(0) * (A - C) + J(0) * (B - C)$$
$$P1 = P0 + \Delta 01I * (A - C) + \Delta 01J * (B - C)$$
$$P2 = P0 + \Delta 02I * (A - C) + \Delta 02J * (B - C)$$
$$P3 = P0 + \Delta 03I * (A - C) + \Delta 03J * (B - C)$$

P0	P1
P2	P3

P0 is computed at 20x24 mantissa precision and P1 to P3 are computed at 8x24 mantissa precision. So far no visual degradation of the image was seen using this scheme.

Multiplies (Full Precision): 2
Multiplies (Reduced precision): 6
Subtracts 19x24 (Parameters): 2
Adds: 8

FORMAT OF P0's IJ: Mantissa 20 Exp 4 for I + Sign
Mantissa 20 Exp 4 for J + Sign

FORMAT of Deltas (x3):Mantissa 8 Exp 4 for I + Sign
Mantissa 8 Exp 4 for J + Sign

Total number of bits for one quad worth of IJ data: 20*2 + 8*6 + 4*8 + 4*2 = 128
All numbers are kept using the un-normalized floating point convention: if exponent is different than 0 the number is normalized if not, then the number is un-normalized. The maximum range for the IJs (Full precision) is +/- 63 and the range for the Deltas is +/- 127.

7. Shader Limitations

The sequencer unit and compiler need to be aware of a series of limitations in the shader functionality. These are limitations on the pixel shader functionality as well as on the vertex shader functionality. In reality, the compiler does not need to pay attention to these limitations since sequencer will detect them and react accordingly. However, for compiler optimization reasons, we describe here the various latency issues revolving around our shader pipe implementation.

1) The use of Previous Vector result (PV) and Previous Scalar result (PS) values.
The following sequence is being executed:
ADD R0 = R3, R4.
MUL R2 = R0, R1 and the desired R0 values are the ones coming from the ADD instruction (ie a dependant instruction).
Because of the pipeline latencies involved, the R0 value from the ADD instruction won't be available in GPRs until one instruction later from the moment the MUL instruction enters the execution pipeline. The sequencer will write the same sequence as follows:
ADD R0 = R3, R4.
MUL R2 = PV, R1

Alternatively, if wanted, the compiler can force the use of the PV register by instead using the following instruction:
MUL R2 = PV, R1 (instead of MUL R2 = R0, R1).

The following sequence is being executed:
ADD R0.x = R3, R4.
MUL R2 = R0, R1 and the desired R0 values is the one coming from the ADD instruction.
Because of the pipeline latencies involved, the R0.x value from the ADD instruction won't be available in GPRs until one cycle later from the moment the MUL instruction enters the execution pipeline. The compiler can introduce a NOP instruction in between ADD and MUL, but it does not have to. The sequencer will detect this dependency case and insert a MOV PV, PV on the vector side and a MOV PS, PS on the scalar side as well (MOV PV,PV is the HW translation of a NOP).

As a conclusion: If the Previous Vector Result is used explicitly in the code, then the instruction will be executed as is. If a dependant use of a masked register is done instead, the sequencer is going to introduce a NOP between the two instructions in order to achieve the right behavior.

2) There is a one-instruction load-use delay between a MOVA instruction and use of a 'indexed' constant. If this delay cannot be honored, the MOVA instruction should be followed by a MOV instruction.

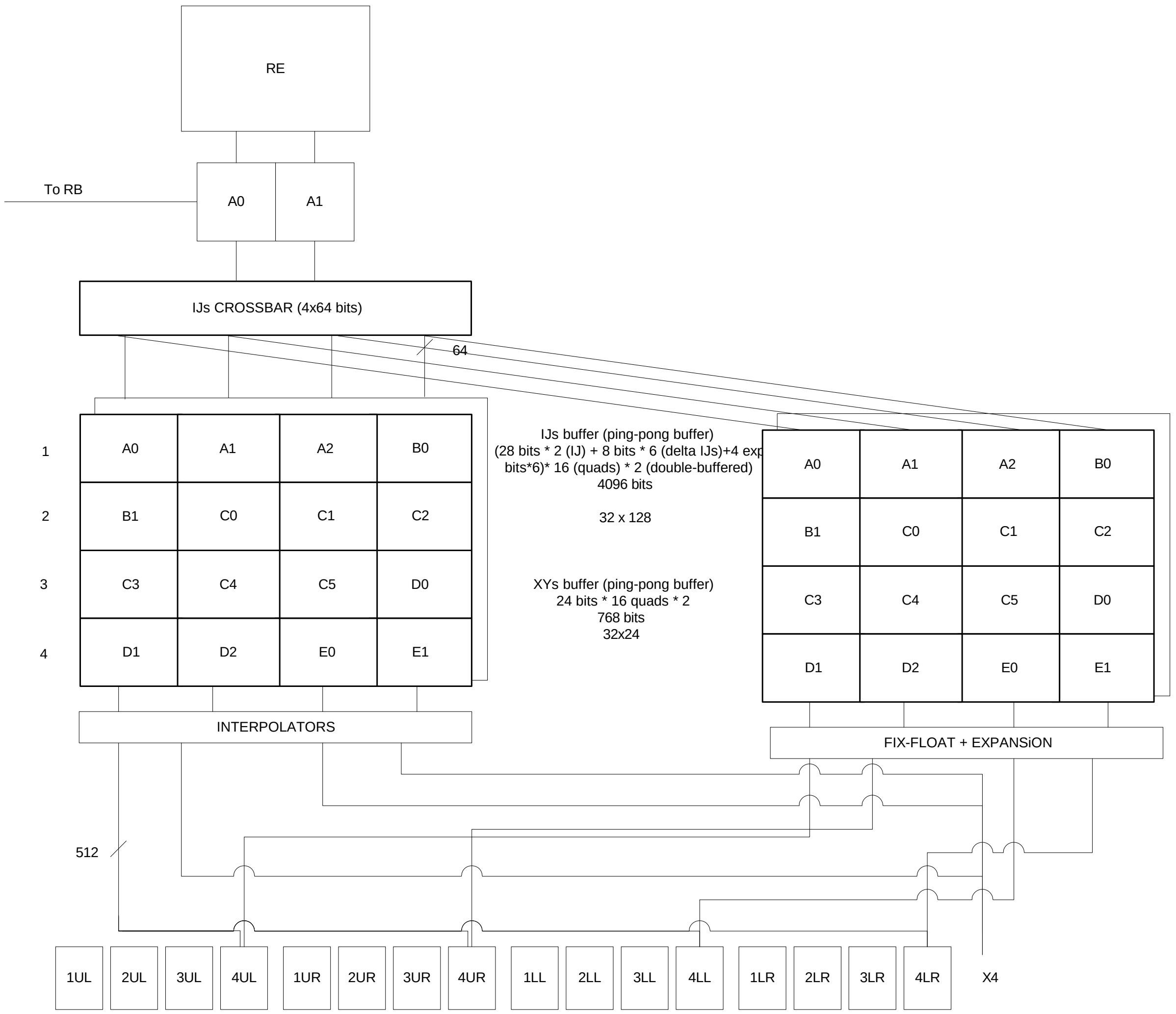
3) General Coissue limitations. A scalar instruction may not be coissued with a 3-argument vector instruction.

8. Hardware Implementation Specifics

8.1 General Information on the Shader Floating Point arithmetic

Two special cases of floating-point numbers are recognized: zero is defined as any number with a zero exponent and infinity (or NaN) is defined as any number with an exponent of 255. The mantissa is cleared for both input and output values of zero and infinity while the sign bit is cleared for input and output zeros. The result of a multiplication involving zero is always defined as zero. The result of any addition involving infinity is always defined as infinity, with the added stipulation that any addition involving negative infinity always results in negative infinity. The Vector Engine/Scalar Engine works with standard IEEE floating-point numbers although all math operations are performed without rounding.

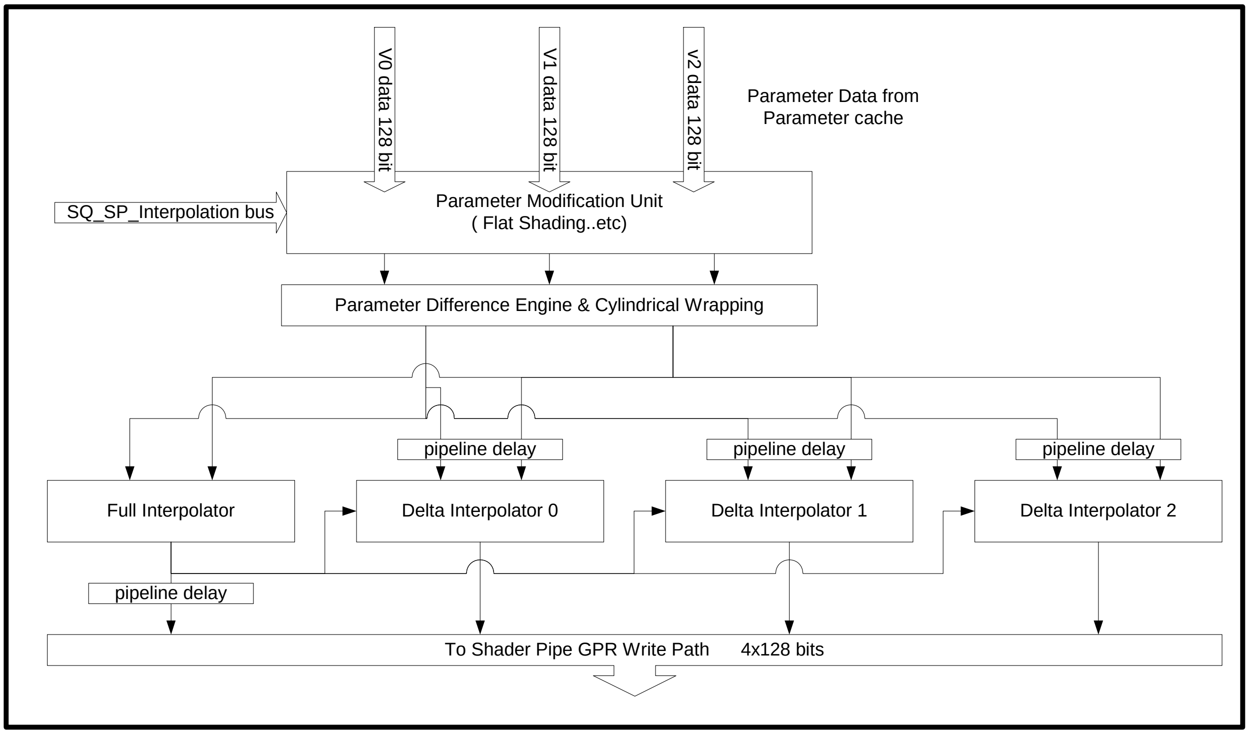
8.2 Interpolators and IJ/XY Buffers



The above diagram represents a high level description of IJ and XY fifos, as well as Parameter Interpolation unit (Interpolators). The double buffer on the left represents the IJ fifo. Each of the cells in that buffer represents a quad of pixels worth of IJ interpolation data. The double buffer on the right represents the XY Address fifo. There is 512 bits of data transferred from the interpolators into each shader pipe (GPRs) per cycle, a totall of 2048 bits across the whole chip. The reading and writing of IJ/XY fifos is controlled by the sequencer via the “SQ to SP: Intepolator bus” interface described in the Section 5.1.4 of this document.

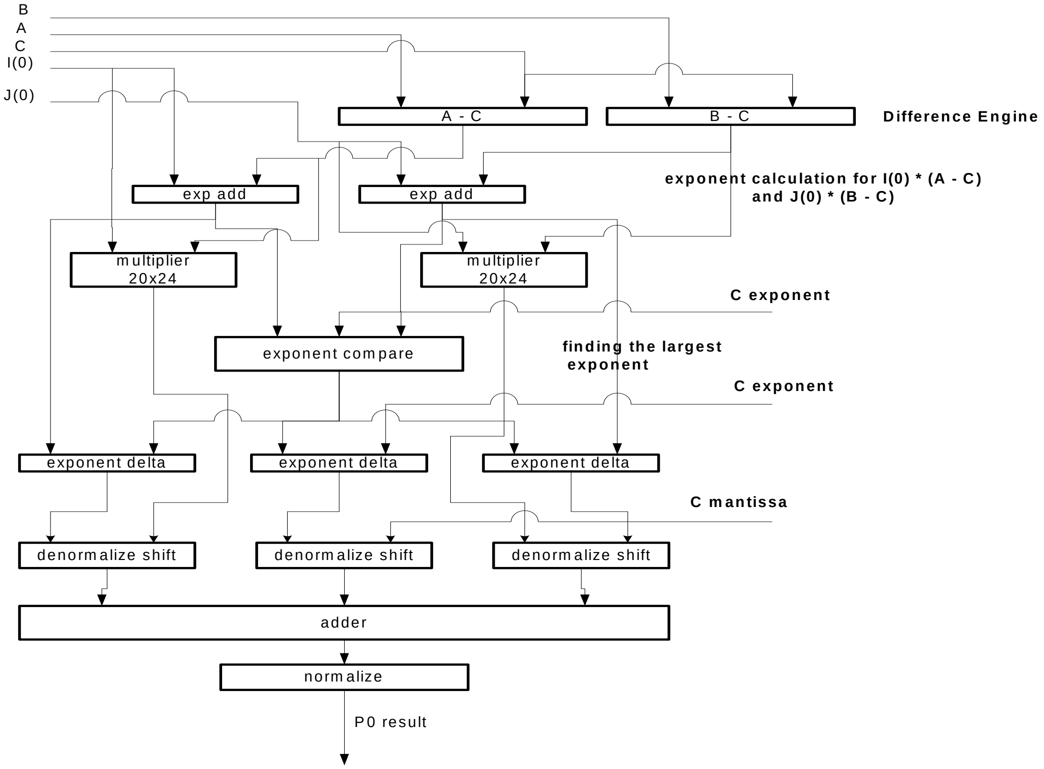
8.2.1 *Interpolators*

The following diagram describes the interpolation unit for one shader pipe. There are four instances of the same in the R400 architecture. Parameter Difference Engine calculates the difference between the values of a given parameter at the vertices. These deltas are inputs into the four interpolators shown below. IJ intepolation data from the IJ fifos would be the other required input into the interpolators. The first interpolation unit (High Precision Interpolator) is used to calculate the interpolated values for Pixel 0, the upper left pixel of the quad. The result of this interpolator is then fed into the lower precision interpolator (Delta Interpolators) used to calculate the parameter values for the other three pixels of the quad. In Section 6 of this document you will find the mathematical description of the barycentric interpolation equations.

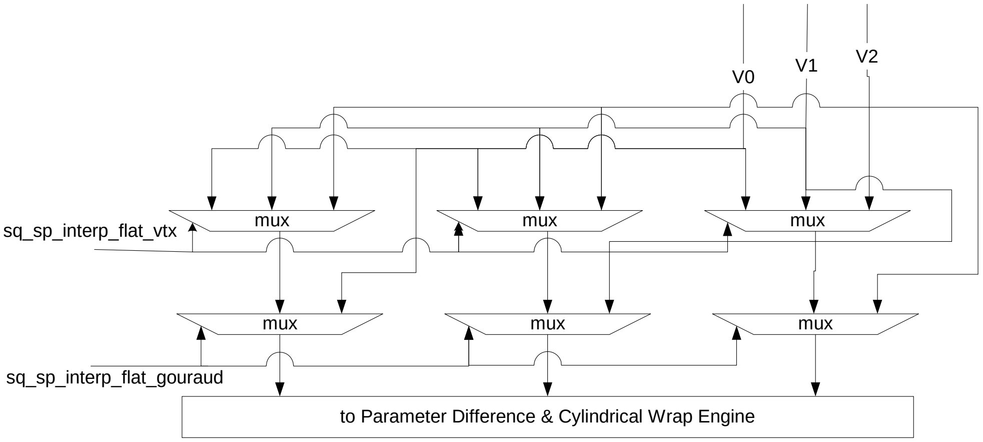


8.2.1.1 Interpolation Units

The following diagram describes the data path for a high precision interpolation unit for one 32-bit channel of the parameter data. The diagram does not exactly describe the hardware implementation of the interpolator. In order to simplify the diagram, three denormalize shifters are shown. In reality, only two out of three mantissa terms about to be added, need to be denormalized to the third mantissa term with the largest exponent. The “exponent compare” unit finds the largest exponent of the following three terms: C, I(0) (A-C) and J(0)*(B-C).



8.2.1.2 Parameter Selection Unit


V0
V1
V2
mux
mux
mux
sq_sp_interp_flat_vtx
mux
mux
mux
sq_sp_interp_flat_gouraud
to Parameter Difference & Cylindrical Wrap Engine

The Parameter Modification unit described in the above diagram is used to preprocess the vertex parameter data coming from the Parameter Cache unit before they enter the Difference engine. It's in this block that the selection of the provoking vertex parameter value, for flat shading, is done. The pertaining control signals are part of the "SQ to SP: Interpolator bus" interface described in Section 5.1.5 of this document.

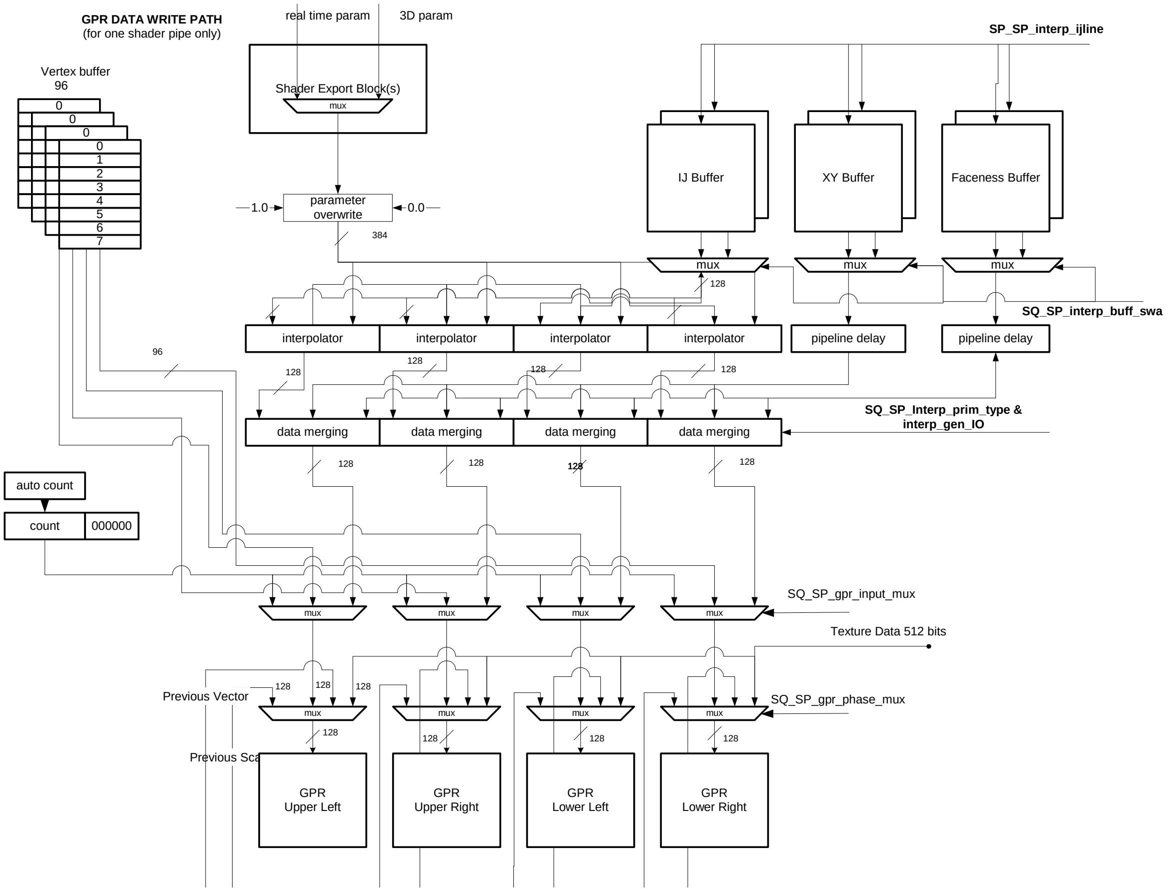
8.2.1.3 Parameter Difference & Cylindrical Wrap Engine

The purpose of the unit is to calculate the delta differences between the parameter values at vertices. These deltas are then used as input values into the interpolator units. Also, it's in this unit that cylindrical wrap adjustment of the parameter values is done. All the functions are implemented on per channel basis. One alternative solution would be for the delta differences between the parameter values to be done in the SX blocks. This way we can cut down from 4 channels * 2 substracts/channel * 4 pipes = 32 subtractors to 8 if placed in SX blocks. This may complicate the Cylindrical Wrap implementation, which in itself requires a bunch of subtracts. The control/state signals are part of the "SQ_SP: Interpolator Bus" interface. Please refer to Section 5.1.5 of this document for a detailed definition of this control interface.

8.2.2 *GPR Write Path*

The diagram below shows all the possible data paths going into the GPR write paths, their selection and routing.
The drawing shows four GPR units representing any four GPR units of a shader pipeline that are in the same phase.
From the pipeline-timing point of view, interpolated parameters of pixels belonging to the same quad are flowing through the data paths described in the drawing at any given time. As it was mentioned before, the interpolators have two sets of inputs, the IJ data coming out of the IJ buffers and vertex parameter data coming from the parameter caches via the SX (Shader Export) blocks. The SX blocks are responsible for multiplexing between the real time parameters and vertex parameter data coming from the parameter caches. The outputs from the interpolators get merged into 128 vectors with data coming from XY and Faceness buffers. The next level of muxing controlled by SQ_SP_gpr_input_mux part of the "SQ_SP: Interpolation bus" interface is used to route between vertex data/indices and interpolated pixel parameters. The next and last level of multiplexing is used to multiplex the writing into GPRs of texture fetch return data, interpolated pixel data, previous scalar and previous vector result.
The truth table below describes all the possible "merge" combinations between the interpolated, XY and Faceness data based on the **SQ_SP_interp_prim_type** and **SQ_SP_interp_gen_IO** signals found in "SQ_SP: Interpolation bus" interface. The above signals are also used to control the overwrite of the parameter values coming from the parameter cache with constants 0.0 and 1.0 when doing expansion for point sprite primitive types.

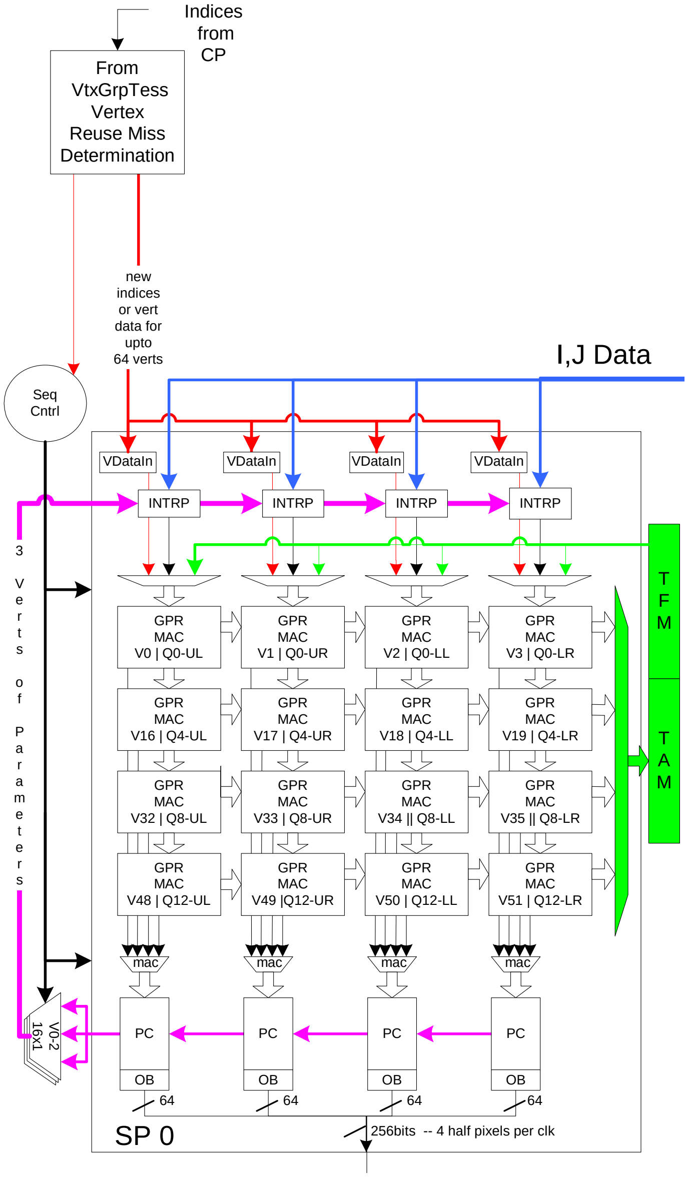
SQ_SP_interp_gen_IO	**SQ_SP_param_type[2]**	**Merge Logic Result**
0	0	Interpolated data
0	1	Interpolated data
1	0	X, Y, don't care, don't care
1	1	X, Y, S, T



8.3 Vector Unit

8.3.1 *Vector Unit Pipeline*

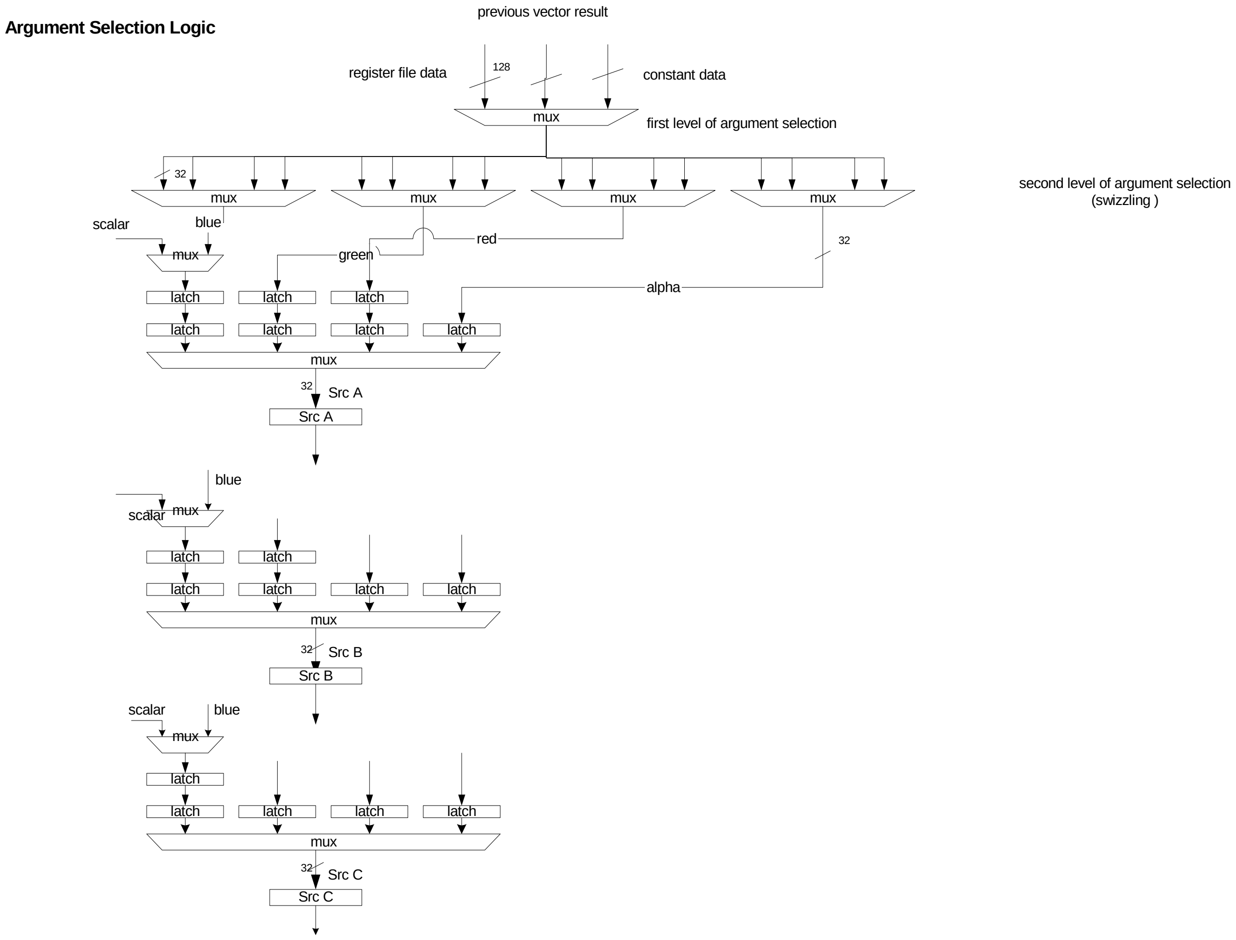
The diagram below describes a complete set of the Vector Units present in a single shader pipeline with interpolator units at the top of the pipeline and the parameter caches at the bottom of it.



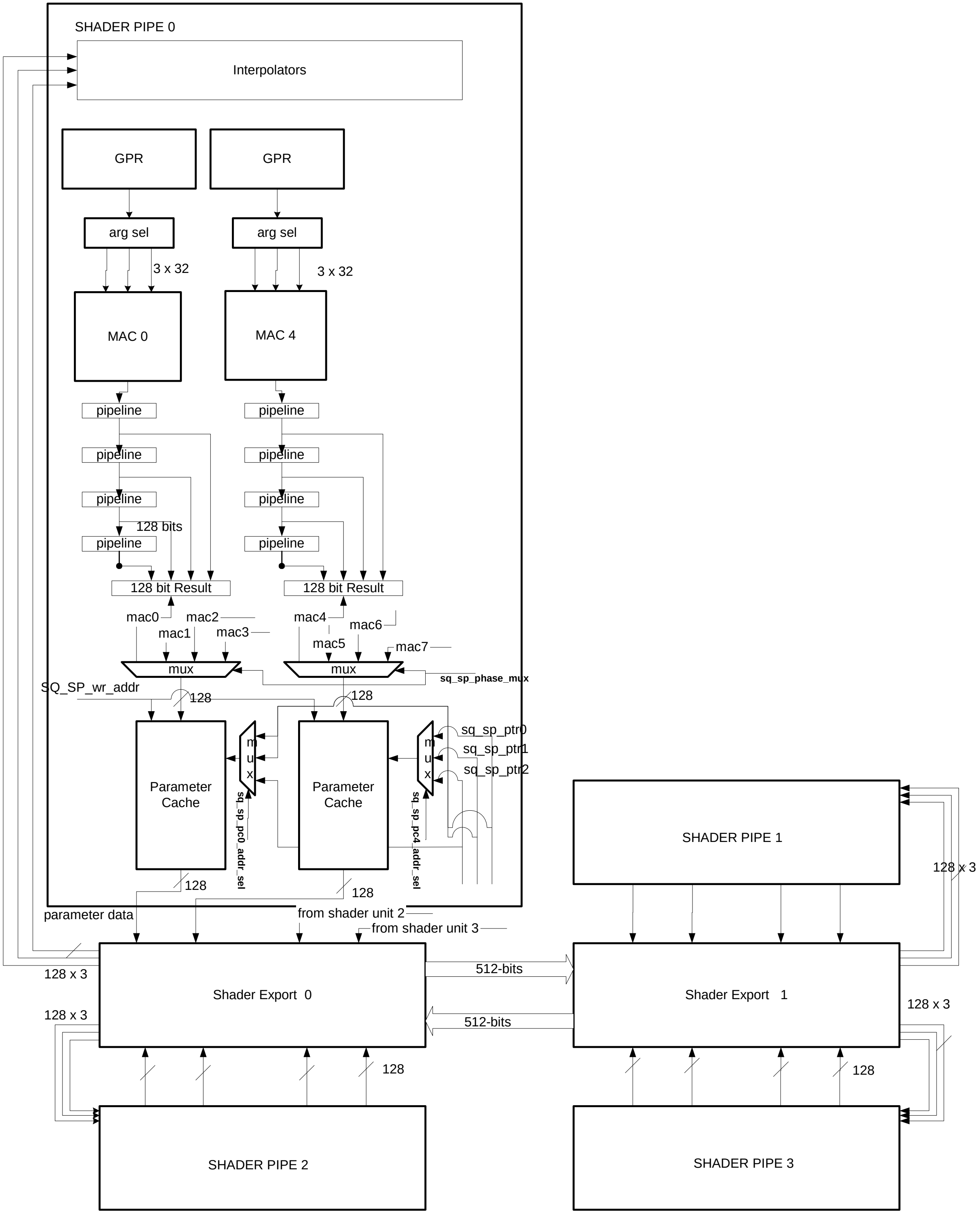
The control of all the multiplexers present at the input and output of the GPRs, input of the TAM and output of the parameter caches is done by the sequencer. In most cases this control represents the phase cycle relative to the ALU instruction start. In the above diagram, the GPR/MAC units of the same row are synchronous in phase. From one row to the other the execution sequence is phased out by one cycle.

8.3.2 *Argument Selection and Routing*

The following diagram describes the routing and selection logic for the ALU instruction operands. Select fields or combination of select fields present in the ALU instruction word controls the first two levels of argument selection. The multiplexing control logic into the SRC (A, B or C) registers is a 4-state FSM that serializes the red, green, blue and alpha input arguments into the MAC unit. As mentioned before, the instruction ALU word specifies a pointer into the GPR register file for each of the sources. Also, the instruction specifies a set of select bits for the final MAC input operands A, B and C. The possible choices that Src A, B and C arguments can be selected from are: Register File Data, Constant Data, Previous Vector Result and Previous Scalar Result. The first level of muxing selects between Previous Vector Result, Register File Data and Constant Data, all 128-bit values. The second level of muxing implements the swizzling logic at 32-bit channel granularity.



8.3.3 *Parameter Data Path*



The output of the vertex shader program, transformed parameter data is written into Parameter Cache memories. There is one parameter cache (128x128) for each vector unit, resulting in 4 parameter cache memories for one shader pipe. The write data path of each parameter cache is time-multiplexed between the 4 MAC units of a given shader vector unit. The 7-bit write address into parameter cache memory comes from Sequencer unit. This address is broadcasted to all the shader pipes. For more details on the parameter cache write path, please refer to Section 5.1.7 of this document.

The read address into parameter cache memories is a result of a muxing of three possible 7-bit address pointers broadcasted by the Sequencer to all shader pipes. These three pointers are part of “Parameter Cache Read Control Bus” described in Section 5.1.6 of this document.

There are 512-bit worth of data transferred from Shader Pipe to SX blocks for every read of the parameter cache.

Once read from the parameter caches, the parameter data is then routed by the SX units into the interpolation units at the top of the shader pipe.

8.4 Scalar Unit

A large portion of R400 Scalar Unit specification is based on the R300 Math Unit specifications as described in “R300 Vertex Assembler & Processor Architecture Specification” document Version 0.93. Section 3.3 of this document describes all the math operations and their respective opcodes supported by the Scalar Unit. There are four Scalar Units present in each shader pipe. The Scalar unit can perform one action each cycle with the result appearing after a latency of eight cycles. In this respect, the Scalar engine is fully pipelined.

Scalar Engine Functions and Precision

Function	Range	Precision[1]
x^n	$0.0 \le x \le 1.0$ $-128 \le n \le 128$	7 ??
$\frac{1}{x}$	$-\infty \le x \le \infty$	23
$\frac{1}{\sqrt{x}}$	$-\infty \le x \le \infty$	23
e^x	$-128.0 \le x \le 128.0$	16 ??
2^x	$0.0 \le x \le 1.0$	23
$\log_2(x)$	$1.0 \le x \le 2.0$	23

[1]Precision loses 1 bit each time range is doubled

The Scalar Engine also calculates $\frac{1}{x}$, $\frac{1}{\sqrt{x}}$, $\log_2(x)$, and 2^x using sixteen 32-entry lookup tables with four unsigned multipliers (2 ea. 16x16, and 2 ea. 12x12) and a 26-bit unsigned adder. These functions also use a 24-bit floating-point multiplier that applies the power term for the x^n and e^x functions (maybe we do not need this !!??).

8.4.1 *Scalar Engine Pipeline*

All the opcodes (except MULTIPLY) that do not use the power function pipeline are processed in the high precision pipeline. The mantissa for all functions is determined using the following function:

$$P = f(x_0) + A(x - x_0) + B(x - x_0)^2 + C(x - x_0)^3$$

The terms $f(x_0)$, A , B, and C were determined at 32 evenly spaced sample points over the range shown in Table 9-1 for each of the functions $\frac{1}{x}$, $\frac{1}{\sqrt{x}}$, $\log_2(x)$, and 2^x. These terms are stored in sixteen tables, four for each function, and addressed by bits 22 – 18 of the input value x (bits 23 - 19 for the $\frac{1}{\sqrt{x}}$ function). The $(x - x_0)$ terms are deltas which allow third-order interpolation between the sample points. The resolution of these lookup table values for each of the four functions is shown below. The largest resolution is assumed in the hardware. Lookup table values are left justified and smaller terms are packed with zeros. Deltas are also left justified, but smaller deltas have the truncated bits OR'ed with their LSB before and after being squared & cubed. The product terms are truncated to the indicated precision and then right justified before being added to the $f(x_0)$ *term* .

High Precision Function Coefficient Resolution

Function	$f(x_0)$	A	B	C
$\frac{1}{x}$	24	20	15	10
$\frac{1}{\sqrt{x}}$	24	20	16	12
$\log_2(x)$	25	20	15	9
2^x	25	20	14	7
Hardware	25	20	16	12

8.4.1.1.1.1.1 High Precision Pipeline Exp$_2$ Preprocessing

The two opcodes EXP_BASE2_DX and EXP_BASE2_FULL_DX require that the input value be preprocessed before it enters the main pipeline. The input exponent is first checked to determine whether the value can be represented within the IEEE single precision format (see Power Function Pipeline Log$_2$ to Real Conversion for details). If the log$_2$ exponent falls within the valid range, then the exp^2 preprocessor converts the input value to a S7.23 format value:

Input Mantissa (24 bits): `1mmmmmmmmmmmmmmmmmmmmmmm`

Input Exponent (biased): `eeeeeeee`

Bias: `01111111`

Subtract to get *Unbiased (signed) Exponent* : `SUUUUUUUU`

Absolute Value of *Unbiased Exponent* to get *shift value*: `vvvvv`

Shift Input Mantissa by *shift value* to get *adjusted input* (30 bits):

If *Unbiased Exponent* Positive, shift left: `aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa`

If *Unbiased Exponent* Negative, shift right: `aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa`

Twos complement if input is negative to get S7.23 value: `sbbbbbbbccccccccccccccccccccccc`

8.4.1.1.1.1.2 High Precision Pipeline Mantissa Calculation

Calculation of the high precision mantissa is the same regardless of function type. Exceptions involve the selection of input values and are discussed in the process flow that follows. The multiplication of the first order term *(A * (x – x0))* uses the power function pipeline multiplier. All other multiplications use dedicated multlipliers.

<u>Get Starting value</u>:

EXP opcode:

Use preprocessed S7.23 value: `0sbbbbbbbccccccccccccccccccccccc`

All other opcodes:

Use iAG_ME_IN_A, an IEEE floating point value: seeeeeeeemmmmmmmmmmmmmmmmmmmmmmm

Get Index:

RECIP opcode:

Use LSB of exponent and 4 MSB's of input mantissa: ....iiiii...

All other opcodes:

Get lookup table index from 5 MSB's of input mantissa: ....iiiii..

Get *delta*:

RECIP & **RECIPSQRT** opcodes:

Use 19 LSB's of input mantissa for *delta*: ddddddddddddddddddd

All other opcodes:

Use 18 LSB's of input mantissa for *delta*: dddddddddddddddddd0

Get *Slope1:*

Term *A* from Lookup Table: AAAAAAAAAAAAAAAAAAAA

delta: ddddddddddddddddddd

Multiply to get *mult0*: qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq

RECIP, RECIPSQRT opcodes:

Use 20 MSB's of *mult0* for *Slope1*: qqqqqqqqqqqqqqqqqqqq0

All other opcodes:

Use 21 MSB's of *mult0* for *Slope1*: qqqqqqqqqqqqqqqqqqqqq

Get *ddelta*:

RECIPSQRT & **RECIP** opcodes:

OR together 4 LSB's of *delta*: l

Concatenate bits 18 – 4 of *delta* to get *ddelta*: dddddddddddddddl

RECIP, LOG opcodes:

OR together 5 LSB's of *delta*: l

Concatenate bits 18 – 5 of *delta* to get *ddelta*: ddddddddddddddl0

EXP opcode:

OR together 6 LSB's of *delta*: l

Concatenate bits 18 – 6 of *delta* to get *ddelta*: dddddddddddddl00

Get *delta_square*:

Square *ddelta* to get *mult1*: rrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrr

RECIPSQRT opcode:

OR together bits 16 - 15 of *mult1*:	l
Concatenate bits 31 – 17 of *mult1* to get *delta_square*:	rrrrrrrrrrrrrrrl
RECIP & LOG opcodes:	
OR together bits 17 - 16 of *mult1*:	l
Concatenate bits 31 – 18 of *mult1* to get *delta_square*:	rrrrrrrrrrrrrrl0
EXP opcode:	
OR together bits 18 - 17 of *mult1*:	l
Concatenate bits 31 – 19 of *mult1* to get *delta_square*:	rrrrrrrrrrrrrl00

<u>Get *Slope2*</u>:

Term *B* from Lookup Table:	BBBBBBBBBBBBBBBB
delta_square:	rrrrrrrrrrrrrrrr
Multiply to get *mult2*:	ssssssssssssssssssssssssssssssss

RECIPSQRT opcode:	
Use 16 MSB's of *mult2* for *Slope2*:	sssssssssssssss0
RECIP opcode:	
Use 15 MSB's of *mult2* for *Slope2*:	0ssssssssssssss0
EXP opcode:	
Use 14 MSB's of *mult2* for *Slope2*:	000sssssssssssss
LOG opcode:	
Use 15 MSB's of *mult2* for *Slope2*:	00ssssssssssssss

<u>Get *dddelta*</u>:

RECIPSQRTopcode:	
OR together 8 LSB's of *delta*:	l
Concatenate bits 18 – 8 of *delta* to get *dddelta*:	dddddddddddl
RECIP opcode:	
OR together 10 LSB's of *delta*:	l
Concatenate bits 18 – 10 of *delta* to get *dddelta*:	dddddddddl00
EXP opcode:	
OR together 13 LSB's of *delta*:	l
Concatenate bits 18 – 13 of *delta* to get *dddelta*:	ddddddl00000
LOG opcode:	
OR together 11 LSB's of *delta*:	l
Concatenate bits 18 – 11 of *delta* to get *dddelta*:	ddddddddl000

<u>Get *ddelta_square*</u>:

RECIPSQRT opcode:

OR together bits 20 - 19 of *mult1*: `l`

Concatenate bits 31 – 21 of *mult1* to get *ddelta_square*: `rrrrrrrrrrl`

RECIP opcode:

OR together bits 22 - 21 of *mult1*: `l`

Concatenate bits 31 – 23 of *mult1* to get *ddelta_square*: `rrrrrrrrrl00`

EXPopcode:

OR together bits 25 - 24 of *mult1*: `l`

Concatenate bits 31 – 26 of *mult1* to get *ddelta_square*: `rrrrrrl00000`

LOG opcode:

OR together bits 23 - 22 of *mult1*: `l`

Concatenate bits 31 – 24 of *mult1* to get *ddelta_square*: `rrrrrrrrl000`

<u>Get *delta_cubed*</u>:

ddelta : `dddddddddddd`

ddelta_square : `rrrrrrrrrrrr`

Multiply to get *mult3*: `tttttttttttttttttttttttt`

RECIPSQRT opcode:

OR together bits 12 - 11 of *mult3*: `l`

Concatenate bits 23 – 13 of *mult3* to get *delta_cubed*: `tttttttttttl`

RECIP opcode:

OR together bits 14 - 13 of *mult3*: `l`

Concatenate bits 23 – 15 of *mult3* to get *delta_cubed*: `tttttttttl00`

EXP opcode:

OR together bits 17 - 16 of *mult3*: `l`

Concatenate bits 23 – 18 of *mult3* to get *delta_cubed*: `ttttttl00000`

LOG opcode:

OR together bits 15 - 14 of *mult3*: `l`

Concatenate bits 23 – 16 of *mult3* to get *delta_cubed*: `ttttttttl000`

<u>Get *slope3*</u>:

Term *C* from Lookup Table: `CCCCCCCCCCCC`

delta_cubed: `tttttttttttt`

Multiply to get *mult4*: `uuuuuuuuuuuuuuuuuuuuuuuu`

RECIPSQRT opcode:

Use 12 MSB’s of *mult4* for *Slope3*: `uuuuuuuuuuuu0`

RECIP opcode:	
Use 10 MSB’s of *mult4* for *Slope3*:	`00uuuuuuuuuu0`
EXP opcode:	
Use 7 MSB’s of *mult4* for *Slope3*:	`000000uuuuuuu`
LOG opcode:	
Use 9 MSB’s of *mult4* for *Slope3*:	`0000uuuuuuuuu`

Term *f(x$_0$)* from Lookup Table:	`fffffffffffffffffffffffff`
Slope1:	`qqqqqqqqqqqqqqqqqq`
Slope2:	`ssssssssssssss`
Slope3:	`uuuuuuuuuu`
Add to get *Full Mantissa*:	`MMMMMMMMMMMMMMMMMMMMMMMMM`
Right shift 2 bits to get *Final Mantissa*:	`MMMMMMMMMMMMMMMMMMMMMMM`

8.4.1.1.1.1.3 *High Precision Pipeline Log$_2$ Post Processing*

The two opcodes LOG_BASE2_DX and LOG_BASE2_FULL_DX post process the *full mantissa* to produce the final mantissa and exponent. This step is the same as the process performed in the power function pipeline except that the precision is higher.

Full Mantissa:	`MMMMMMMMMMMMMMMMMMMMMMMMM`
Unbiased Input Exponent :	`SUUUUUUU`
Combine *Unbiased Exponent* with *Final Mantissa*:	`SUUUUUUUMMMMMMMMMMMMMMMMMMMMMMMMM`
Sign multiply to get *Log$_2$ Mantissa*:	`0ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ`
Normalize (using left *shift L*) to get *Norm Mantissa*:	`NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN`
Biased Exponent for *shift L* of 0:	`10000110`
Shift L:	`LLL`
Subtract to get Biased *Log$_2$ Exponent*:	`EEEEEEEE`
Norm Mantissa (right shift by 2, drop MSB):	`NNNNNNNNNNNNNNNNNNNNNNN`
Biased Log$_2$ Exponent:	`EEEEEEEE`
Unbiased Exponent Sign:	`S`
Combine to get final Log$_2$ Conversion Result:	`SEEEEEEEENNNNNNNNNNNNNNNNNNNNNNN`

8.4.1.1.1.1.4 *High Precision Pipeline Exponents*

Calculation of the exponents for the LOG_BASE2_DX, LOG_BASE2_FULL_DX, EXP_BASE2_DX, and EXP_BASE2_DX opcodes is covered in the corresponding pre and post processing sections. Exponents for the RECIP_DX, RECIP_FF, RECIP_SQRT_DX, and RECIP_SQRT_FF are calculated in the following way:

Base Exponent:

Input Exponent (biased):	`0eeeeeeee`
RECIPSQRT opcode:	
Use Input Exponent (biased):	`0bbbbbbbb`
RECIP opcode:	
Invert Input Exponent (biased):	`1bbbbbbbb`

Exponent Adjustment:

RECIPSQRT opcode:	
Input is power of 4 (bits 23 – 0 are zero):	`101111111`
Input is not power of 4:	`101111101`
RECIP opcode:	
Input is power of 2 (bits 22 – 0 are zero):	`111111111`
Input is not power of 2:	`111111110`

Final Exponent:

Base Exponent	`bbbbbbbbb`
Exponent Adjustment:	`1x11111xx`
Add to get *Intermediate Exponent* :	`iiiiiiiiz`
RECIPSQRT opcodes:	
Use bits 8 – 1 of intermediate exponent:	`iiiiiiii`
RECIP opcode:	
Use bits 7 – 0 of intermediate exponent:	`iiiiiiiz`

8.4.1.1.1.1.5 High Precision Special Outputs

The Math Engine computes or pipelines several values required by the DirectX 8.0 specification. These values include partial terms for the EXP and LOG opcodes, and the diffuse lighting attribute for the LIT opcode. The EXP opcode requires the input power to be split into integer and fractional parts, with the integer part then raised to a binary power. The integer and fractional parts are readily available in the form of the S7.23 fixed-point value found in the EXP_2 preprocessing step. The integer value is already in binary power form and can be muxed onto the oVE_WDATA output (with a zero mantissa). The fractional part of the power term needs to be normalized and converted to floating point format. Since that is a function that is performed in the LOG_2 post processing circuit, the lower 23 bits of the S7.23 value is left shifted 10 bits and muxed into that circuit. The LOG opcode requires that the input term be split into exponent and mantissa parts, with the exponent converted into a floating-point value. The mantissa is easily combined with an exponent of zero (or 0x7f biased) and muxed onto the oVE_WDATA output. Converson of the exponent requires an additional fixed-to-float circuit to generate the result since the LOG_2 post processing circuit is already used to convert the primary output for that opcode. The diffuse lighting attribute for the LIT opcode is simply pipelined from the iAG_ME_IN_B input and muxed onto the oVE_WDATA output.

8.4.1.1.1.1.6 Determination of High Precision Coefficients

The coefficients used to calculate the high precision mantissa were determined using third-order Lagrange polynomials. Since the mantissa is calculated separately from the exponent, the build range for each function was chosen so that the range of the function output values did not change dramatically (see below).

Build Ranges for Lagrangian Coeffiecients

Function	Build Range
$\frac{1}{x}$	x = 1.0 to 2.0
$\frac{1}{\sqrt{x}}$	x = 2.0 to 8.0
2^x	x = 1.0 to 2.0
$\log_2(x)$	x = 2.0 to 4.0

The build ranges were then divided into 32 equal segments. The coefficients for each segment were determined using the two end points of the segment, $f(x_0)$ and $f(x_1)$, as well as two points on the segment itself, $f(x_{01})$ and $f(x_{02})$, such that the distance along the x-axis between any two adjacent points, h, was 1/96 the total length of the build range (see figure below).

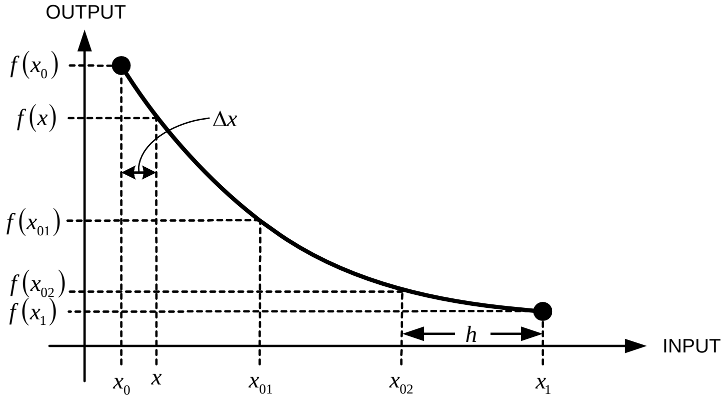


Figure 8-1: Build Range Segment

The third-order Lagrange polynomial used to approximate each segment of each function at any point x along the segment is as follows:

$$\hat{f}(x) = f(x_0)\left[\frac{(x-x_{01})(x-x_{02})(x-x_1)}{(x_0-x_{01})(x_0-x_{02})(x_0-x_1)}\right]+$$
$$f(x_{01})\left[\frac{(x-x_0)(x-x_{02})(x-x_1)}{(x_{01}-x_0)(x_{01}-x_{02})(x_{01}-x_1)}\right]+$$
$$f(x_{02})\left[\frac{(x-x_0)(x-x_{01})(x-x_1)}{(x_{02}-x_0)(x_{02}-x_{01})(x_{02}-x_1)}\right]+$$
$$f(x_1)\left[\frac{(x-x_0)(x-x_{01})(x-x_{02})}{(x_1-x_0)(x_1-x_{01})(x_1-x_{02})}\right]$$

Since $x_{01} = x_0 + h$, $x_{02} = x_0 + 2h$, and $x_1 = x_0 + 3h$, the polynomial can be rewritten in terms of x, x_0, h, $f(x_0)$, $f(x_1)$, $f(x_{01})$, and $f(x_{02})$:

$$\hat{f}(x) = -\frac{1}{6h^3} f(x_0)[(x-(x_0+h))(x-(x_0+2h))(x-(x_0+3h))]+$$
$$\frac{1}{2h^3} f(x_{01})[(x-(x_0))(x-(x_0+2h))(x-(x_0+3h))]+$$
$$-\frac{1}{2h^3} f(x_{02})[(x-(x_0))(x-(x_0+h))(x-(x_0+3h))]+$$
$$\frac{1}{6h^3} f(x_1)[(x-(x_0))(x-(x_0+h))(x-(x_0+2h))]$$

The polynomial is then factored for successive powers of the quantity $(x-x_0)$, i.e., 1, $(x-x_0)$, $(x-x_0)^2$ and $(x-x_0)^3$. Using the notation $\Delta x = x - x_0$, the polynomial can be reduced to the following form:

$$\hat{f}(x) = \sum_{i=0}^{3} a_i (\Delta x)^i$$

where each of the terms a_i is:

$$a_0 = f(x_0)$$

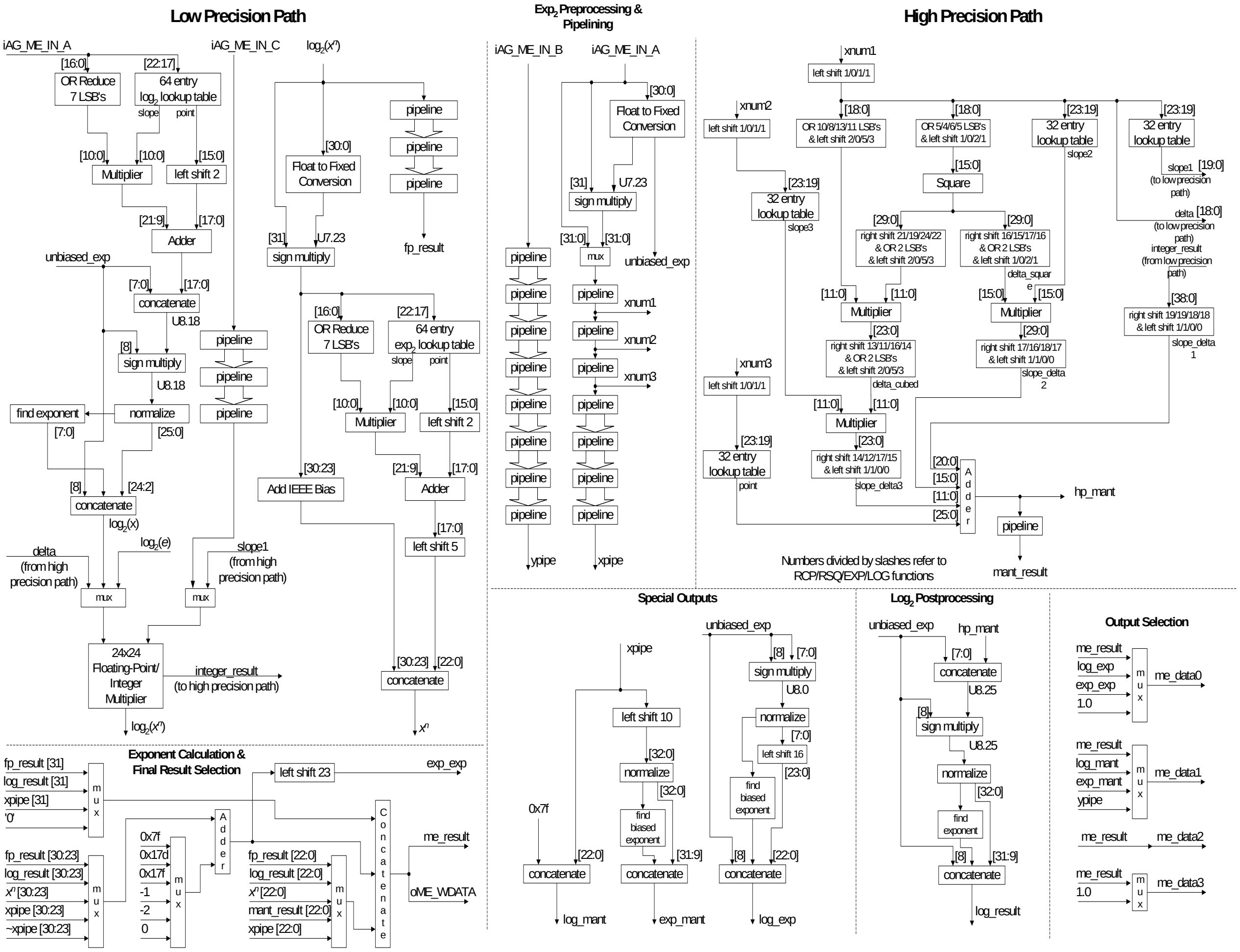
$$a_1 = \frac{1}{3h}[-5.5 f(x_0) + 9 f(x_{01}) - 4.5 f(x_{02}) + f(x_1)]$$

$$a_2 = \frac{1}{9h^2}[9 f(x_0) - 22.5 f(x_{01}) + 18 f(x_{02}) - 4.5 f(x_1)]$$

$$a_3 = \frac{1}{27h^3}[-4.5 f(x_0) + 13.5 f(x_{01}) - 13.5 f(x_{02}) + 4.5 f(x_1)]$$

Using the equations above, coefficients for each segment of each function were calculated and stored with the precision necessary to achieve 24-bit resolution.

The diagram below represents the Scalar Engine part of the pipeline that implements the LUT (Lookup Table) type scalar functions.



9. Open Issues

1. There's still to be decided the final location for difference engine used in calculating the delta between the vertex parameter values. The two possible place are:
    a. Next to the interpolators inside the parameter engines at the top of each shader pipe, but replicating this way the same logic across 4 shader pipelines.
    b. SX (Shader Exports). If this is the case, then Cylindrical Wrap logic should be propably be moved to the SX blocks.

2. The location of the real time parameter caches as well as the routing of this data by SX units into the parameter interpolators.
3. Data expansion logic in the path from the Fetch Engines into shader units/GPRs. Currently, the texture return data into GPRs is defined as a 512-bit bus for each shader pipe (Section 4.3.3).

```
//depot/r400/arch/doc/gfx/RE/R400_Sequencer.doc
... #18 change 10172 edit on 2001/11/16 by llefebvr@llefebvre_laptop_r400 (binary+l)

        chikin in order to move to documents to the new branch

... #17 change 9346 edit on 2001/11/06 by llefebvr@llefebvre_laptop_r400 (binary+l)

        sequencer spec backup

... #16 change 8606 edit on 2001/10/26 by llefebvr@llefebvre_laptop_r400 (binary+l)

        Sequencer spec V1.0.

... #15 change 8175 edit on 2001/10/22 by llefebvr@llefebvre_laptop_r400 (binary+l)

        sequencer v1.0 BACKUP ONLY not yet complete.

... #14 change 8081 edit on 2001/10/19 by llefebvr@llefebvre_laptop_r400 (binary+l)

        One before last major architectural revision of the sequencer before the implementation
spec. Control flow is complete and was accepted by SW team. Remains before freezing 1.0 :
external and internal interfaces.

... #13 change 7930 edit on 2001/10/17 by llefebvr@llefebvre_laptop_r400 (binary+l)

        version 0.8 of the sequencer spec. It contains the new control flow porcedure as well as
updated external interfaces.

... #12 change 7380 edit on 2001/10/05 by llefebvr@llefebvre_laptop_r400 (binary+l)

        version 0.7 of the sequencer. Interfaces and control managment added.

... #11 change 7261 edit on 2001/10/03 by llefebvr@llefebvre_laptop_r400 (binary+l)

        backup of the sequencer + register loading diagram

... #10 change 6865 edit on 2001/09/24 by llefebvr@llefebvre_laptop_r400 (binary+l)

        new spec of the Sequencer.

... #9 change 6790 edit on 2001/09/21 by llefebvr@llefebvre_laptop_r400 (binary+l)

        RE spec backup + HZ stats + SC spec backup

... #8 change 5698 edit on 2001/08/24 by llefebvr@llefebvre_laptop_r400 (binary+l)
```

```
        version 0.4 of the sequencer

... #7 change 5289 edit on 2001/08/13 by llefebvr@llefebvre_laptop_r400 (binary+l)

        added an exemple of registry file management

... #6 change 5260 edit on 2001/08/13 by llefebvr@llefebvre_laptop_r400 (binary+l)

        updated spec for sequencer

... #5 change 4001 edit on 2001/07/05 by llefebvr@llefebvre_laptop_r400 (binary+l)

        lockin is on

... #4 change 4000 edit on 2001/07/05 by llefebvr@llefebvre_laptop_r400 (binary)

        sequencer checkin

... #3 change 3999 edit on 2001/07/05 by pmitchel@pmitchel_iris (binary+l)

        change file type to lock

... #2 change 3330 edit on 2001/06/07 by llefebvr@llefebvre_laptop_r400 (binary)

        safety backup

... #1 change 3105 add on 2001/05/25 by llefebvr@llefebvre_laptop_r400 (binary)

        backup sequencer
```

```
//depot/r400/doc_lib/design/blocks/sq/R400_Sequencer.doc
... #62 change 118771 edit on 2003/08/29 by llefebvr@llefebvr_r400_montreal (binary+l)

        Fixing number of bits in the auto-count.

... ... branch into //depot/r600/r400_doc_lib/design/blocks/sq/R400_Sequencer.doc#1
... ... branch into //depot/yamato/legacy/r400/doc_lib/design/blocks/sq/R400_Sequencer.doc#1
... #61 change 114724 edit on 2003/08/04 by llefebvr@llefebvr_r400_montreal (binary+l)

        Corrected the max number for mem exports to be 5 instead of 9.

... #60 change 109954 edit on 2003/07/09 by llefebvr@llefebvr_r400_montreal (binary+l)

        Fixing VC table.

... #59 change 107253 edit on 2003/06/20 by llefebvr@llefebvr_r400_montreal (binary+l)

        Backup, no major changes.

... #58 change 103310 edit on 2003/05/30 by llefebvr@llefebvr_r400_montreal (binary+l)

        Added comments about address register.

... #57 change 101984 edit on 2003/05/21 by llefebvr@llefebvre_laptop_r400 (binary+l)

        more precisions on XYST generated register.

... #56 change 101037 edit on 2003/05/14 by llefebvr@llefebvr_r400_montreal (binary+l)

        Fixing the spec some more to match R500. Added some diagrams (SQ internals)

... #55 change 100004 edit on 2003/05/08 by llefebvr@llefebvr_r400_montreal (binary+l)

        Some interface updates.

... #54 change 98760 edit on 2003/05/02 by llefebvr@llefebvr_r400_montreal (binary+l)

        forgot to remove 1 waterfall signal.

... #53 change 98500 edit on 2003/05/01 by llefebvr@llefebvr_r400_montreal (binary+l)

        Refreshing the interfaces per Andi's last mail.

... #52 change 98401 edit on 2003/04/30 by llefebvr@llefebvr_r400_montreal (binary+l)

        Updated the SQ->SP interfaces for the R500.
```

```
... #51 change 97450 edit on 2003/04/24 by llefebvr@llefebvre_laptop_r400 (binary+l)

        Updated stall conditions.
        Made swizzle changes.
        Added more R500 specifics.

... #50 change 97161 edit on 2003/04/23 by llefebvr@llefebvre_laptop_r400 (binary+l)

        interface name changes for the SQ->SP fetch swizzles.

... #49 change 97092 edit on 2003/04/23 by llefebvr@llefebvre_laptop_r400 (binary+l)

        Added SP stall conditions to the SQ spec.

... #48 change 96572 edit on 2003/04/19 by llefebvr@llefebvr_r400_montreal (binary+l)

        Documentation changes for R500.

... #47 change 93456 edit on 2003/04/02 by llefebvr@llefebvr_r400_montreal (binary+l)

        update to the control flow instruction.
        Adding timing diagram for the SQ->VC/TP transfers.

... #46 change 92587 edit on 2003/03/28 by llefebvr@llefebvre_laptop_r400 (binary+l)

        added swizzle codes to the spec.

... #45 change 87762 edit on 2003/02/28 by llefebvr@llefebvr_r400 (binary+l)

        Added the new SX interface.

... #44 change 82838 edit on 2003/02/07 by llefebvr@llefebvr_r400 (binary+l)

        small update regarding the implementation change for the Pos Allocated / PC allocated
bits.

... #43 change 81538 edit on 2003/02/03 by llefebvr@llefebvr_r400 (binary+l)

        Missed bits 41 and 42 in the SQ_EXEC instruction format. Those are RESERVED as
well.

... #42 change 81401 edit on 2003/02/03 by llefebvr@llefebvr_r400 (binary+l)

        refined the interfaces to the SP to specify wich signals should or shouldn't be pipelined.
```

```
... #41 change 80832 edit on 2003/01/30 by llefebvr@llefebvre_laptop_r400 (binary+l)

        wording change for the predicate override bit.

... #40 change 78984 edit on 2003/01/23 by llefebvr@llefebvr_r400 (binary+l)

        small correction on memory export buffer sizes.

... #39 change 77093 edit on 2003/01/16 by llefebvr@llefebvr_r400 (binary+l)

        Modified the alloc instruction to include a no-serial bit.

... #38 change 77001 edit on 2003/01/15 by llefebvr@llefebvr_r400 (binary+l)

        Interface change from the SX (alloc dealloc bus) and interface change from the SP
(predicates and kill mask).

... #37 change 74942 edit on 2003/01/07 by llefebvr@llefebvre_laptop_r400 (binary+l)

        New revision of the spec.

... #36 change 59941 edit on 2002/10/29 by llefebvr@llefebvre_laptop_r400 (binary+l)

        backup and SQ->SP interface change.

... #35 change 58563 edit on 2002/10/22 by llefebvr@llefebvre_laptop_r400 (binary+l)

        Defined the memory exports better.

... #34 change 57527 edit on 2002/10/16 by llefebvr@llefebvre_laptop_r400 (binary+l)

        Clarifications and minor updates. Version 2.08.

... #33 change 56881 edit on 2002/10/14 by llefebvr@llefebvre_laptop_r400 (binary+l)

        Loops, jumps and calls are now using a 13 bit address which allows to jump and call and
loop around any control flow addresses (does not requires to be even anymore).

... #32 change 56604 edit on 2002/10/11 by llefebvr@llefebvre_laptop_r400 (binary+l)

        Revision 2.06 of the spec.

... #31 change 50456 edit on 2002/09/10 by llefebvr@llefebvre_laptop_r400 (binary+l)

        New spin of the SQ spec including some interface changes and auto-counter architectural
changes (for multipass pixel/vertex shaders).
```

```
... #30 change 49717 edit on 2002/09/05 by llefebvr@llefebvre_laptop_r400 (binary+l)

        updated spec.

... #29 change 49244 edit on 2002/09/03 by llefebvr@llefebvre_laptop_r400 (binary+l)

        new spec

... #28 change 48839 edit on 2002/08/29 by llefebvr@llefebvre_laptop_r400 (binary+l)

        Updated the SQ->SP interface. Added comment on the Constant load bus.

... #27 change 48092 edit on 2002/08/26 by llefebvr@llefebvre_laptop_r400 (binary+l)

        Added the new SQ->SP instruction interface.

... #26 change 46138 edit on 2002/08/14 by llefebvr@llefebvre_laptop_r400 (binary+l)

        Changed the MASK mnemonics to KILL.
        Added DST opcode.
        Added MUL_PREV2 opcode.
        Reordered the opcodes in primlib and SP.
        Implemented the new KILL and SET SCALAR opcodes, they are now all comparing the
ALPHA channel to 0.0f (instead of comapring against the RED channel).

... #25 change 44021 edit on 2002/08/02 by llefebvr@llefebvre_laptop_r400 (binary+l)

        New parameter generation scheme included in the spec.

... #24 change 41600 edit on 2002/07/19 by llefebvr@llefebvre_laptop_r400 (binary+l)

        SQ backup.

... #23 change 40691 edit on 2002/07/15 by llefebvr@llefebvre_laptop_r400 (binary+l)

        New sequencer spec.

... #22 change 37951 edit on 2002/07/03 by llefebvr@llefebvre_laptop_r400 (binary+l)

        Backup

... #21 change 31433 edit on 2002/06/03 by llefebvr@llefebvre_laptop_r400 (binary+l)

        Backup and minor updates.
```

... #20 change 27717 edit on 2002/05/13 by llefebvr@llefebvre_laptop_r400 (binary+l)

Changed CF opcodes, SQ->SP interface and SP->SQ constant index load interface.

... #19 change 26057 edit on 2002/05/02 by llefebvr@llefebvre_laptop_r400 (binary+l)

Modification on the Control flow instructions.

... #18 change 23957 edit on 2002/04/19 by llefebvr@llefebvre_laptop_r400 (binary+l)

The new control flow scheme is now included in v2.0 of the sequencer spec.

... #17 change 23946 edit on 2002/04/19 by llefebvr@llefebvre_laptop_r400 (binary+l)

Last version of the spec with the old control flow scheme

... #16 change 20320 edit on 2002/03/25 by llefebvr@llefebvre_laptop_r400 (binary+l)

Upated the interfaces and added an exporting rule section.

... #15 change 20199 edit on 2002/03/22 by llefebvr@llefebvre_laptop_r400 (binary+l)

Some minor changes to the SQ interfaces.

... #14 change 19503 edit on 2002/03/18 by llefebvr@llefebvre_laptop_r400 (binary+l)

Changed the interfaces to reflect the fact that the PCs are now in the SX blocks

... #13 change 17380 edit on 2002/03/04 by llefebvr@llefebvre_laptop_r400 (binary+l)

New revision of the sequencer spec.

... #12 change 14466 edit on 2002/02/04 by llefebvr@llefebvre_laptop_r400 (binary+l)

Version 1.7 of the Sequencer spec.

... #11 change 13058 edit on 2002/01/15 by llefebvr@llefebvre_laptop_r400 (binary+l)

redondant opcodes corrected.

... #10 change 13057 edit on 2002/01/15 by llefebvr@llefebvre_laptop_r400 (binary+l)

There was a small error in the control flow section. Checked in the spec so that Richard has a correct version to build the assembler on.

... #9 change 12708 edit on 2002/01/09 by llefebvr@llefebvre_laptop_r400 (binary+l)

new revision of the sequencer spec v1.6

... #8 change 12335 edit on 2002/01/03 by llefebvr@llefebvre_laptop_r400 (binary+l)

backup of the spec

... #7 change 11833 edit on 2001/12/17 by llefebvr@llefebvre_laptop_r400 (binary+l)

backup

... #6 change 11503 edit on 2001/12/11 by llefebvr@llefebvre_laptop_r400 (binary+l)

Version 1.5 of the sequencer spec. See Revision changes of the document for details.

... #5 change 11443 edit on 2001/12/10 by llefebvr@llefebvre_laptop_r400 (binary+l)

backup. Updated the constant memory management section by copying stuff from the R400 state management document by Mike Mantor.

... #4 change 11393 edit on 2001/12/07 by llefebvr@llefebvre_laptop_r400 (binary+l)

backup.

... #3 change 11306 edit on 2001/12/06 by llefebvr@llefebvre_laptop_r400 (binary+l)

New revision of the sequencer spec.

... #2 change 11226 edit on 2001/12/05 by llefebvr@llefebvre_laptop_r400 (binary+l)

Updated the register spec.

... #1 change 11061 branch on 2001/12/03 by pmitchel@pmitchel_r400_win_marlboro (binary+l)

mv doc_lib/parts to doc_lib/design/blocks

... ... branch from //depot/r400/doc_lib/parts/sq/R400_Sequencer.doc#1,#3
//depot/r400/doc_lib/parts/sq/R400_Sequencer.doc
... #3 change 11048 edit on 2001/12/03 by llefebvr@llefebvre_laptop_r400 (binary+l)

submited for Paul to move stuff around again.

... ... branch into //depot/r400/doc_lib/design/blocks/sq/R400_Sequencer.doc#1
... #2 change 10774 edit on 2001/11/27 by llefebvr@llefebvre_laptop_r400_emu (binary+l)

opened the files with the wrong client

... #1 change 10705 branch on 2001/11/26 by pmitchel@pmitchel_r400_win_marlboro (binary+l)

another rename to match r300

... ... branch from //depot/r400/doc_lib/parts_lib/sq/R400_Sequencer.doc#3
//depot/r400/doc_lib/parts_lib/sq/R400_Sequencer.doc
... #3 change 10699 branch on 2001/11/26 by pmitchel@pmitchel_r400_win_marlboro (binary+l)

doing rename properly

... ... branch from //depot/r400/doc_lib/blocks/sq/R400_Sequencer.doc#5
... ... branch into //depot/r400/doc_lib/parts/sq/R400_Sequencer.doc#1
... #2 change 10698 delete on 2001/11/26 by pmitchel@pmitchel_r400_win_marlboro (binary+l)

fix mistake

... #1 change 10691 branch on 2001/11/26 by pmitchel@pmitchel_r400_win_marlboro (binary+l)

rename "blocks" to "parts_lib"

... ... branch from //depot/r400/doc_lib/blocks/sq/R400_Sequencer.doc#1,#2
//depot/r400/doc_lib/blocks/sq/R400_Sequencer.doc
... #5 change 10697 add on 2001/11/26 by pmitchel@pmitchel_r400_win_marlboro (binary+l)

recover

... ... branch into //depot/r400/doc_lib/parts_lib/sq/R400_Sequencer.doc#3
... #4 change 10695 delete on 2001/11/26 by pmitchel@pmitchel_r400_win_marlboro (binary+l)

rename

... #3 change 10693 edit on 2001/11/26 by llefebvr@llefebvre_laptop_r400 (binary+l)

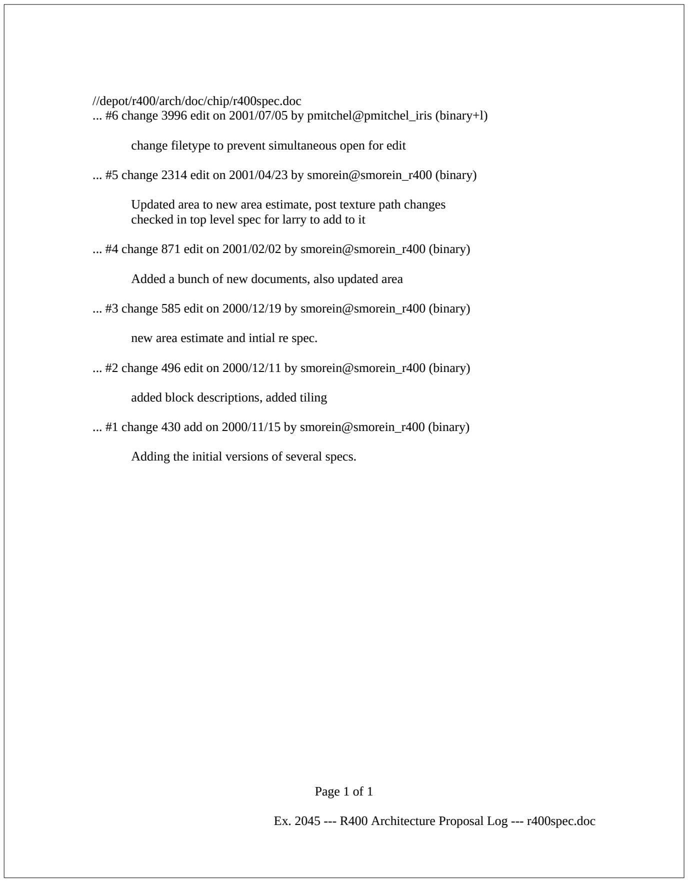
closing for Paul to move files around

... #2 change 10676 edit on 2001/11/26 by llefebvr@llefebvre_laptop_r400 (binary+l)

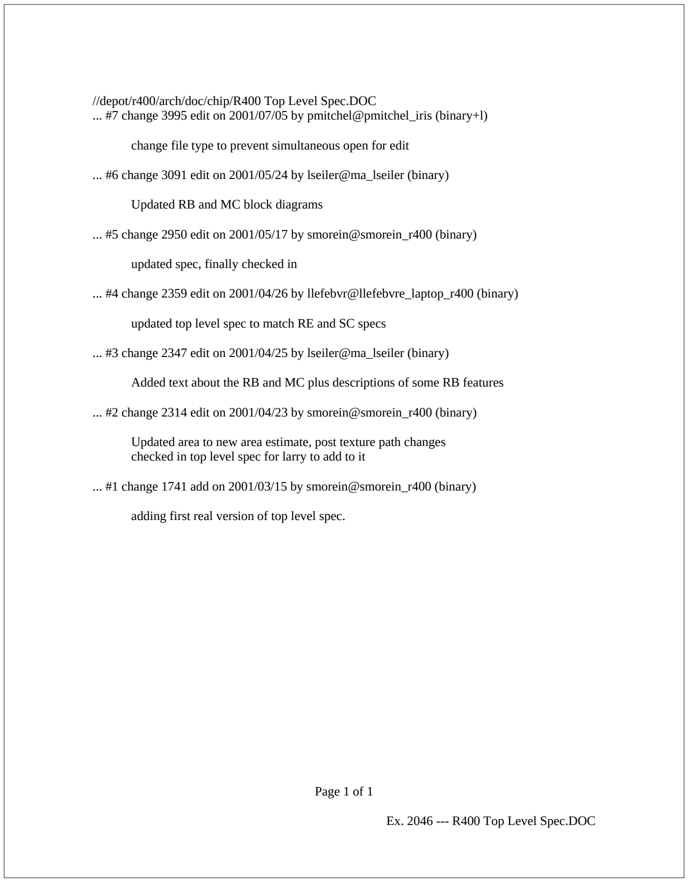
changed the file type to binary and locked

... ... branch into //depot/r400/doc_lib/parts_lib/sq/R400_Sequencer.doc#1
... #1 change 10674 add on 2001/11/26 by llefebvr@llefebvre_laptop_r400 (binary)

new spin on the sequencer spec



```
//depot/r400/arch/doc/chip/r400spec.doc
... #6 change 3996 edit on 2001/07/05 by pmitchel@pmitchel_iris (binary+l)

        change filetype to prevent simultaneous open for edit

... #5 change 2314 edit on 2001/04/23 by smorein@smorein_r400 (binary)

        Updated area to new area estimate, post texture path changes
        checked in top level spec for larry to add to it

... #4 change 871 edit on 2001/02/02 by smorein@smorein_r400 (binary)

        Added a bunch of new documents, also updated area

... #3 change 585 edit on 2000/12/19 by smorein@smorein_r400 (binary)

        new area estimate and intial re spec.

... #2 change 496 edit on 2000/12/11 by smorein@smorein_r400 (binary)

        added block descriptions, added tiling

... #1 change 430 add on 2000/11/15 by smorein@smorein_r400 (binary)

        Adding the initial versions of several specs.
```



```
//depot/r400/arch/doc/chip/R400 Top Level Spec.DOC
... #7 change 3995 edit on 2001/07/05 by pmitchel@pmitchel_iris (binary+l)

        change file type to prevent simultaneous open for edit

... #6 change 3091 edit on 2001/05/24 by lseiler@ma_lseiler (binary)

        Updated RB and MC block diagrams

... #5 change 2950 edit on 2001/05/17 by smorein@smorein_r400 (binary)

        updated spec, finally checked in

... #4 change 2359 edit on 2001/04/26 by llefebvr@llefebvre_laptop_r400 (binary)

        updated top level spec to match RE and SC specs

... #3 change 2347 edit on 2001/04/25 by lseiler@ma_lseiler (binary)

        Added text about the RB and MC plus descriptions of some RB features

... #2 change 2314 edit on 2001/04/23 by smorein@smorein_r400 (binary)

        Updated area to new area estimate, post texture path changes
        checked in top level spec for larry to add to it

... #1 change 1741 add on 2001/03/15 by smorein@smorein_r400 (binary)

        adding first real version of top level spec.
```

//depot/r400/doc_lib/design/blocks/sp/Shaders.doc
... #36 change 131864 edit on 2003/11/13 by frising@frising_r400_win_marlboro (binary+l)

-For cube instruction SrcA swizzle is now .zzxy. Also tried to clarify the differences between what's shown in the numerics doc and what actually happens in the HW for cube instruction.

... ... branch into //depot/r600/r400_doc_lib/design/blocks/sp/Shaders.doc#1
... ... branch into //depot/yamato/legacy/r400/doc_lib/design/blocks/sp/Shaders.doc#1
... #35 change 126058 edit on 2003/10/10 by frising@frising_r400_win_marlboro (binary+l)

-update scalar mova instructions to return MAX_S(SrcC)

... #34 change 125972 edit on 2003/10/09 by frising@frising_r400_win_marlboro (binary+l)

-update vector mova instruction to be two operand with result to GPR being max of operands. Scalar mova instructions were updated to always return srcC.w.

... #33 change 111385 edit on 2003/07/16 by frising@frising_r400_win_marlboro (binary+l)

v.1.99
-add scalar sin and cos instructions

... #32 change 108338 edit on 2003/06/27 by frising@frising_r400_win_marlboro (binary+l)

v.1.98
-show scalar instructions SUB_CONST_0 and SUB_CONST_1 as negated adds to be consistent with other subtraction instructions.

... #31 change 107630 edit on 2003/06/24 by frising@frising_r400_win_marlboro (binary+l)

v.1.97
-update cube instruction to take two operands. Output produced in a different order too.

... #30 change 99777 edit on 2003/05/07 by frising@frising_r400_win_marlboro (binary+l)

v.1.96
-update rules for masking with Color/Fog export.

... #29 change 99452 edit on 2003/05/06 by frising@frising_r400_win_marlboro (binary+l)

v.1.95
-Update spec to show that masking of exports is allowed for all exports now (possible only exception fog - TBD).

... #28 change 97033 edit on 2003/04/22 by frising@frising_r400_win_marlboro (binary+l)

v.1.94
-update mova* instructions to return SrcA (like a mov) on the vector side and SrcC.W replicated on the scalar side.
-update pred* instructions to only use W channel of operands.
-update GPR write-back table to show that scalar component is used when both scalar and vector write masks are enabled.

... #27 change 78304 edit on 2003/01/21 by frising@ma_frising (binary+l)

v.1.93
-Z export from pixel sahder now in X channel.
-updated mova, kill and predicate instructions coissue rules now that we have a separate bus for mova results.
-add note saying input modifiers do not apply to PreviousScalar.
-show not used opcodes.
-add clamping to mova result sent to SQ.
-add 6 new scalar instructions that operate on a constant and GPR and associated documentation.

... #26 change 74194 edit on 2003/01/02 by frising@ma_frising (binary+l)

v.1.92

-MUL_PREV2 scalar instruction now checks if PreviousScalar or SrcC.X is a NaN and returns -MAX_FLOAT if so.

... #25 change 68144 edit on 2002/12/03 by frising@ma_frising (binary+l)

v.1.91

-FLOOR opcode was not producing correct results for negative 'integer' inputs.

Changed from:
FLOOR:

```
If (SrcA < 0.0f)
  Result = TRUNC(SrcA) + -1.0f;
Else
  Result = TRUNC(SrcA)
```

To:
FLOOR:

```
Result = TRUNC(SrcA)
If ( (SrcA < 0.0f) && (SrcA != Result) )
```

```
  Result += -1.0f;
```

-Note that emulator was calling math library floor function so this should not be an emulation issue.

... #24 change 67933 edit on 2002/12/02 by frising@ma_frising (binary+l)

v.1.90

-While we don't allow CLI relative addressing into the export file on r400, future chips based on r400 may. This check-in moves the export masking behavior bit (bit[6] of vector destination pointer) to bit[14] (bit[6] of scalar destination pointer). Bit[6] of vector destination pointer now controls logical vs. CLI relative addressing into the register or export file. When exporting, this is a Must Be Zero (logical) field for for software on r400. This will provide binary compatibility.

-Software will need to coordinate this change with emulator release.

... #23 change 64418 edit on 2002/11/15 by frising@ma_frising (binary+l)

v.1.89

-show FRACT instructions being implemented as: SRC + -FLOOR(SRC)
-This should now sync the instructions with v.0.99b of numerics doc.

... #22 change 64381 edit on 2002/11/15 by frising@ma_frising (binary+l)

v.1.88

-specify that Result.X of CUBE instruction returns 2.0f * MajorAxis instead of max to avoid confusion. See numerics doc for details of CUBE instruction.

-specify function of all instructions in WZYX order. Nothing changing here; it's just for spec consistency (and happens to reflect the actual order of operations in the shader pipe hardware).

... #21 change 63732 edit on 2002/11/13 by frising@ma_frising (binary+l)

v.1.87

-when using absolute constant addressing all constants in instruction are absolute. This allows compiler to easily perform mov operations on absolute constants.

-Document instruction word in WZYX order (i.e. high bits to low bits). Documentation issue only.

-lots more work cleaning up instruction word documentation (please study).

-It may not have been clear in the past but indexing of exports is not permitted. This update should make that obvious.

-removed following export restriction which no longer applies: '1) When doing a Scalar export of 'pixels' or 'position', only the 'W' component will contain the scalar result. The other 3 components will be expanded to 0.0. When exporting to 'parameters' the scalar result is put into all 4 components.'

Users should just use the scalar and vector destination masks appropriately to achieve whatever result they want.

-when exporting, bit 6 in instruction word now controls masking behavior during parameter exports when both scalar and vector masks for a component are 0. See table 3.2.1.4

... #20 change 63509 edit on 2002/11/12 by frising@ma_frising (binary+l)

v.1.86

-add note that Constant0 refers to the first constant in the instruction while Constant1 and Constant2 refer to the second and third constants in the instruction respectively.

-added note that the GPR write-back table rules only apply when the scalar and vector destination pointers are the same. This should be obvious, but clarity never hurts.

-when doing an export and both scalar and vector channels are masked a 0.0f is now generated.

-updated Exports Types and Addresses section to match what is in SQ doc.

-tried to clean-up export rules section.

-removed previous vector and previous scalar from source selects in instruction word.

-added clamping code to LOG_CLAMPED instruction.

-fixed a bunch of typos and other misc clean-up including a couple places where I was mixing ABGRs with my WZYXs in the instruction definitions. Tried to be consistent when refering to bits in instruction word.

... #19 change 61151 edit on 2002/11/01 by frising@ma_frising (binary+l)

v.1.85

```
        -clarifed how constant and register addressing works.  For nomenclature I settled on
absolute, logical and relative addressing.  New tables added.
        -documented how constant2 works.
        -Wrote a blurb and added a table on GPR write-back precedence.
        -Cleaned up ALU instruction format table

... #18 change 60264 edit on 2002/10/30 by frising@ma_frising (binary+l)

        v.1.8
        -bring up to date with v.0.99a of R400numerics.doc.  Includes adding new instructions,
updating existing instructions, and moving to counter based predicate scheme.

... #17 change 49786 edit on 2002/09/05 by askende@andi_r400_docs (binary+l)

        new rev. 1.7

... #16 change 49232 edit on 2002/09/02 by askende@andi_r400_docs (binary+l)

        fixed a typo on MUL_PREV2

... #15 change 49141 edit on 2002/08/31 by askende@andi_r400_docs (binary+l)

        new rev of the spec ...rev.1.6

... #14 change 47336 edit on 2002/08/21 by askende@andi_r400_docs (binary+l)

        modified the instruction interface from SQ to SP

... #13 change 46099 edit on 2002/08/14 by askende@andi_r400_docs (binary+l)

        new rev of the spec.

... #12 change 45512 edit on 2002/08/12 by askende@andi_r400_docs (binary+l)

        new rev. 1.4

... #11 change 29310 edit on 2002/05/21 by askende@andi_r400_docs (binary+l)

        fixed typo

... #10 change 14237 edit on 2002/01/30 by askende@andi_r400_docs (binary+l)

        new rev of the document.

... #9 change 13514 edit on 2002/01/22 by askende@andi_r400_docs (binary+l)
```

```
        new rev 1.2

... #8 change 13474 edit on 2002/01/21 by askende@andi_r400_docs (binary+l)

        new rev. 1.1

... #7 change 13471 edit on 2002/01/21 by askende@andi_r400_docs (binary+l)

        new rev 1.1 of the shader pipe

... #6 change 13235 edit on 2002/01/17 by askende@andi_r400_docs (binary+l)

        new rev.

... #5 change 13231 edit on 2002/01/17 by askende@andi_r400_docs (binary+l)

        new shader rev of the shader spec checked in.

... #4 change 13227 edit on 2002/01/17 by askende@andi_r400_docs (binary+l)

        new rev of the shader spec

... #3 change 11895 edit on 2001/12/17 by askende@andi_r400_docs (binary+l)

        new updates of the spec with regards to ALU instruction word definition,
        scalar opcode list and the hardware definition of the scalar unit.

... #2 change 11479 edit on 2001/12/10 by askende@andi_r400_docs (binary+l)

        new revision

... #1 change 11061 branch on 2001/12/03 by pmitchel@pmitchel_r400_win_marlboro
(binary+l)

        mv doc_lib/parts to doc_lib/design/blocks

... ... branch from //depot/r400/doc_lib/parts/sp/Shaders.doc#1,#4
//depot/r400/doc_lib/parts/sp/Shaders.doc
... #4 change 11047 edit on 2001/12/03 by askende@andi_r400_docs (binary+l)

        more updates

... ... branch into //depot/r400/doc_lib/design/blocks/sp/Shaders.doc#1
... #3 change 10809 edit on 2001/11/27 by askende@andi_r400_docs (binary+l)

        new rev of the spec.
```

```
... #2 change 10770 edit on 2001/11/27 by pmitchel@pmitchel_r400_win_marlboro (binary+l)

        change filetype to +l

... #1 change 10769 branch on 2001/11/27 by pmitchel@pmitchel_r400_win_marlboro (binary)

        mv to doc_lib/parts

... ... branch from //depot/r400/arch/doc/gfx/SP/Shaders.doc#1,#18
//depot/r400/arch/doc/gfx/SP/Shaders.doc
... #18 change 10501 edit on 2001/11/21 by askende@andi_r400_docs (binary)

        opcode update

... ... branch into //depot/r400/doc_lib/parts/sp/Shaders.doc#1
... #17 change 10500 edit on 2001/11/21 by askende@andi_r400_docs (binary)

        updated a couple of opcodes

... #16 change 9723 edit on 2001/11/12 by askende@andi_r400_docs (binary)

        new revision

... #15 change 6889 edit on 2001/09/25 by askende@andi_docs (binary)

        newest version

... #14 change 5466 edit on 2001/08/17 by askende@andi_docs (binary)

        new rev of the spec

... #13 change 4960 edit on 2001/08/01 by askende@andi_docs (binary)

        new rev

... #12 change 4929 edit on 2001/07/31 by askende@andi_docs (binary)

        a new rev

... #11 change 4033 edit on 2001/07/06 by askende@andi_docs (binary)

        update of the specs

... #10 change 3585 edit on 2001/06/20 by askende@andi_docs (binary)
```

```
        new rev

... #9 change 3574 edit on 2001/06/20 by askende@andi_docs (binary)

        new rev

... #8 change 3565 edit on 2001/06/19 by askende@andi_docs (binary)

        new rev

... #7 change 3560 edit on 2001/06/19 by askende@andi_docs (binary)

        new rev

... #6 change 3558 edit on 2001/06/19 by askende@andi_docs (binary)

        another rev

... #5 change 3553 edit on 2001/06/19 by askende@andi_docs (binary)

        another rev (rev.03) of the shader spec

... #4 change 3138 edit on 2001/05/29 by askende@andi_docs (binary)

        more updates to the spec

... #3 change 3020 edit on 2001/05/21 by askende@andi_docs (binary)

        another revision of the shader spec

... #2 change 2712 edit on 2001/05/09 by askende@andi_docs (binary)

        more updates

... #1 change 2700 add on 2001/05/09 by askende@andi_docs (binary)

        Shader specifications
```

Change 175294 on 2004/06/24 by mmantor@FL_mmantorLT_r400_win

< initial checkin of osm random generator with minor changes to supporting files>

Change 172110 on 2004/06/07 by llefebvr@llefebvr_r400_linux_marlboro

Fixing bad resource hang on the vertex shader.

Change 171407 on 2004/06/02 by llefebvr@llefebvr_r400_linux_marlboro

Fixing Qnan issue

Change 168815 on 2004/05/19 by llefebvr@llefebvr_r400_linux_marlboro

Integrate of the number of RS stations fixes from Xenos.

Change 168560 on 2004/05/18 by llefebvr@llefebvr_r400_linux_marlboro

Back integration of the AddreValid fix from Xenos.

Change 168033 on 2004/05/14 by llefebvr@llefebvr_r400_linux_marlboro

Integration of Randy's Kill maks changes from Xenos.

Change 167975 on 2004/05/14 by llefebvr@llefebvr_r400_linux_marlboro

Blanket integrate of the Xenos emulator files for SQ,SP,SX to R400.

Change 167630 on 2004/05/13 by mmantor@FL_mmantorLT_r400_win

<made alterations to tp_formatter to work in sp standalone environment made vsp
more hardware like in pred, kill, mova, write enables and update on testbench>

Change 155924 on 2004/03/17 by llefebvr@llefebvr_r400_emu_montreal

Integration from Xenos of the valid bit change. tb_sqspsx tracker data only, no functionnal change here.

Change 155819 on 2004/03/17 by llefebvr@llefebvr_r400_emu_montreal

Integration of the gpr clamping base address change from Xenos.

Change 154760 on 2004/03/12 by kryan@kryan_r400_win_marlboro_XP

Integrate changes from Xenos/devel to my r400/branch to r400/devel

Integrate Xenos/devel to r400 branch

- Added YIELD* keywords to the Shader Assembler.

- Added processing for serial bit in SQ and Shader Assembler.

- Changed handling of Control flow constants in Emulator

to more closely match the hardware.

Change 154141 on 2004/03/10 by llefebvr@llefebvr_r400_linux_marlboro

integration of the valid bit bug fix from Xenos

Change 153494 on 2004/03/08 by llefebvr@llefebvr_r400_emu_montreal

Integration of the VS_DONE_EVENT change from Xenos.

Change 152749 on 2004/03/04 by llefebvr@llefebvre_laptop_r400_emu

Integration of the PRED_COND_CALL change to R400.

Change 152663 on 2004/03/04 by llefebvr@llefebvre_laptop_r400_emu

Fixing COND_JMP and COND_CALL predicated. Also fixing bit mask problem in the SX that controls the tracker.

Change 151381 on 2004/02/26 by llefebvr@llefebvre_laptop_r400_emu

Integration of the cond_pred_execute fix from Xenos

Change 151315 on 2004/02/26 by llefebvr@llefebvr_r400_linux_marlboro

Removing the storage for RT parameters in the emultor from the register space and the SX.

Change 151063 on 2004/02/25 by llefebvr@llefebvr_r400_linux_marlboro

Incorporating Tom's changes to the emulator.

Change 149973 on 2004/02/19 by llefebvr@llefebvr_r400_linux_marlboro

Validated Tom's fix.

Change 149972 on 2004/02/19 by llefebvr@llefebvr_r400_emu_montreal

Fixing the control flow machine in the emulator to remove an extra trip to the RS on a COND_PRED_EXEC opcode.

Change 149638 on 2004/02/17 by llefebvr@llefebvr_r400_emu_montreal

This makes the emulator write to PV/PS on all units even when the operation is predicated. Only the writes to the registers are blocked on predication (and pred vector/kill vector).

Change 149424 on 2004/02/16 by llefebvr@llefebvr_r400_emu_montreal

Fixing REP bug.

Change 149402 on 2004/02/16 by llefebvr@llefebvr_r400_emu_montreal

This fixes the COND_EXEC_PRED opcode so it returns the thread with the correct resource when the predicate vector is dirty.

Change 148881 on 2004/02/12 by llefebvr@llefebvr_r400_emu_montreal

The predicate clean bit was overwritten instead of being "anded" in creating less clause boundaries than it should have on the emulator.

Change 148666 on 2004/02/11 by llefebvr@llefebvr_r400_emu_montreal

This is a fix for the control flow instruction mismatch in the case of a COND_EXEC_PRED opcode. The emulator wasn't updating the resource field while the HW was.

Change 148344 on 2004/02/10 by llefebvr@llefebvr_r400_emu_montreal

Enabling PRED_CLEAN instructions in primlib. Fixed a problem with COND_EXEC_PRED_CLEAN_END opcode in the emulator.

Change 148246 on 2004/02/09 by donaldl@donaldl_xenos_linux_orl

Added flat_shading signal for use in the SX parameter subtract function.
(ie. if flat_shading is true, ignore infinity checks; just do subtract. Result
should be zero.)

Change 148084 on 2004/02/09 by llefebvr@llefebvre_laptop_r400_emu

Implemented the REP function in the emulator.

Change 147964 on 2004/02/07 by donaldl@donaldl_xenos_linux_orl

Fixed bug when calling parameter_sub during creation of vectors
(ie. lessthan0 and greaterpoint5 parameters were swapped).

Change 147952 on 2004/02/07 by mmantor@mmantor_xenos_linux_orl

<added test bench for DiffEng param_sub in the sx for numerical verification>

Change 147697 on 2004/02/06 by llefebvr@llefebvre_laptop_r400_emu

The emulator now ignores the MSB of the count field.

Change 147181 on 2004/02/04 by llefebvr@llefebvre_laptop_r400_emu

Adding Sq->SX event dump file.

Change 146961 on 2004/02/03 by mearl@mearl_xenos_linux_orl

Updated the emulator to match the hardware.

Change 146954 on 2004/02/03 by llefebvr@llefebvre_laptop_r400_emu

Fixing alloc with no serial folowed by serialized instruction bug (emulator wasn't stopping).

Change 146738 on 2004/02/02 by mmantor@FL_mmantorLT_r400_win

<added checking of vector and scalar results all the time, constant address and export control&fog overlay, More values are randomized>

Change 146655 on 2004/01/31 by mmantor@mmantor_xenos_linux_orl

<added include directory for crayola_enum.h and fixed vectpipe makefile to use env variables for dep dir>

Change 146654 on 2004/01/31 by mmantor@FL_mmantorLT_r400_win

<saved environment updates, problem with last checking>

Change 146651 on 2004/01/31 by mmantor@FL_mmantorLT_r400_win

<preparing for intrinsity release, updated makes to be compatible with multiple directories using $ROOT and $BRANCH environment
variables and cleaned up sp_vector.v for readability and removed unused signals, functionally equivalent>

Change 146535 on 2004/01/30 by llefebvr@llefebvr_r400_linux_marlboro

Adding Tom's changes to the emulator and SP testbench.

Change 146499 on 2004/01/30 by llefebvr@llefebvr_r400_linux_marlboro

Removing HW accureate flag

Change 146490 on 2004/01/30 by llefebvr@llefebvr_r400_emu_montreal

Removing numbers.h

Change 146478 on 2004/01/30 by llefebvr@llefebvr_r400_linux_marlboro

Fixing bad casting in the TB

Change 146091 on 2004/01/29 by llefebvr@llefebvr_r400_emu_montreal

Fixing SX buffer size management problem when EXPORT_BUFFER size > 128.

Change 146069 on 2004/01/29 by mmantor@mmantor_xenos_linux_orl

<updated makefile for standalone rand vector generator for vsp>

Change 146063 on 2004/01/29 by mmantor@FL_mmantorLT_r400_win

<added new files for last checkin>

Change 146062 on 2004/01/29 by mmantor@FL_mmantorLT_r400_win

<repartition more vector scaler pipe (vsp) into seperate class to support lower level random test bench and intrinsity efforts.
moved all math from sq_alu.cpp to vsp.cpp>

Change 145619 on 2004/01/27 by llefebvr@llefebvr_r400_emu_montreal

Added VS_FETCH_DONE event and interface pulse to the CP for the PL_LIME_BUFFER early de-allocation of the vertex buffer.

Change 145089 on 2004/01/23 by llefebvr@llefebvr_r400_emu_montreal

Added event propagation to the SX (from the SQ). Need to tie it off in Xenos to the BC.

Change 143941 on 2004/01/20 by llefebvr@llefebvr_r400_linux_marlboro

1) Fixed Kill and waterfall issue (r400sq_killgt_01.cpp)
2) Fixed waterfall and mova co-issue problem (only in non-optimized mode).
3) Fixed SX dumpfile NAN sign propagation problem.

Change 143832 on 2004/01/19 by mmantor@FL_mmantorLT_r400_win

<updated the vectpipetest.exe do process interleaved threads, all opcodes and
changed tb_vector back to simple design. All ops pass 400000 instruction random testing >

Change 143255 on 2004/01/15 by llefebvr@llefebvr_r400_emu_montreal

Added VS_FETCH_DONE control flow instruction.
Added VS_FETCH_DONE pulse to CP interface.
Placed the hooks for when the event will be created.

Change 143096 on 2004/01/14 by hartogs@hartogs_xenos_linux_orl

Change to fix ferret dump file for VGT_SQ_Verts interface. This change passed regress_e and parts_lib_rel.pl.

Change 141868 on 2004/01/09 by llefebvr@llefebvre_laptop_r400_emu

Adding warnings for bad register settings on VS_CONST_BASE and VS_CONST_SIZE (sum must be < 512).

Change 141694 on 2004/01/08 by llefebvr@llefebvre_laptop_r400_emu

Fixing DB_ALU_SIZE to say we need 2X the number of constants than highest use in a state.
Fixing VS_NUM_REG to say it is number of registers -1.
Fixing bug in predicated loop end where the jump back to the start of the loop wasn't taken.

Change 140866 on 2004/01/06 by llefebvr@llefebvre_laptop_r400_emu

This is the new alu code as provided by Tom. uses the new 27 bit adder.
Passed release_parts_lib, the emulator regression and the tb_sqspsx regression. also passed 4000000 vectors on the SP testbench.

Change 140078 on 2003/12/29 by llefebvr@llefebvr_r400_emu_montreal

Fixing flush to zero problem on MUL_PREV2.
Fixing _CONST abs modifier problem.

Change 139972 on 2003/12/29 by llefebvr@llefebvr_r400_emu_montreal

Reverting Tom's changes (try 3).

Change 139970 on 2003/12/29 by llefebvr@llefebvr_r400_emu_montreal

Reverting Tom's changes (try 2).

Change 139968 on 2003/12/29 by llefebvr@llefebvr_r400_emu_montreal

Reverting Tom's changes. They brake the random tests with LSB missmatches.

Change 139340 on 2003/12/23 by llefebvr@llefebvr_r400_emu_montreal

Fixed SX to send Nans on invalid positions.
Fixed waterfalling on _CONST opcodes.
Added more debug info to SX->PA dumps.

Change 139084 on 2003/12/22 by llefebvr@llefebvr_r400_emu_montreal

Adding more dump info in the SX->RB color dump.
Adding bug fixes in 27 bit adder path (not used).

Change 138986 on 2003/12/20 by mmantor@mmantor_xenos_linux_orl

<changed csim to only make one pass for param gen and gen index and write the dump files correctly, fixed a timing loop in pix tthread buffer >

Change 138150 on 2003/12/17 by llefebvr@llefebvr_r400_emu_montreal

Removed power of 2 specific wrapping to allow any size VTX and PIX RS.

Change 138102 on 2003/12/17 by llefebvr@llefebvr_r400_emu_montreal

Added programable VTX and PIX RS sizes (from state register).
Added programable SX export buffer sizes.
Added programable number of EA buffers in SX.
Added programable State export slots in SX.

Change 138048 on 2003/12/17 by llefebvr@llefebvr_r400_emu_montreal

The emulator now outputs an error message and exits uppon bad usage of the address register.

Change 137942 on 2003/12/16 by llefebvr@llefebvr_r400_linux_marlboro

The testbench now compiles and runs on linux.

Change 137864 on 2003/12/16 by rramsey@rramsey_xenos_linux_orl

Change emulator so param cache reads for params not exported by the VS
still show up in sq_sx_pcaddr.
Fix cf_resource_change logic in the cfs so it catches the clause boundary
where a cf instr with only tex instr gets sent to the alu cfs.

Change 137313 on 2003/12/12 by mmantor@FL_mmantorLT_r400_win

<update for standalone vp test>

Change 137273 on 2003/12/12 by llefebvr@llefebvre_laptop_r400_emu

Fixing another data scrmable on TRUNC scalar.
Modified MAX4v to match exactly numerics.
Modified MAX and Min opcodes to remove the subtract that could cause an LSB missmatch on the compare.
Fixed Nan on FLOOR scalar.

Change 137256 on 2003/12/12 by llefebvr@llefebvre_laptop_r400_emu

This is fixing the Nan and data scramble on scalar TRUNC.

Change 137124 on 2003/12/12 by llefebvr@llefebvre_laptop_r400_emu

This is fixing the _CONST opcode failure. This is an SQ only change so it will not affect the SP testbench.

Change 137080 on 2003/12/11 by mmantor@FL_mmantorLT_r400_win

<tmp remove scalar prev opcodes for standalone testing>

Change 137063 on 2003/12/11 by llefebvr@llefebvre_laptop_r400_emu

Changing swizzles on CUBE instruction to exactly match SP.

Change 136833 on 2003/12/10 by llefebvr@llefebvre_laptop_r400_emu

Fixing PRED_SET* opcodes to match HW behavior.
Fixing bad channel select on KILL* and MAX4 opcodes.

Change 136832 on 2003/12/10 by mmantor@FL_mmantorLT_r400_win

<added all ops and provide pred opcode swizzle changes and kill in the standalone vectpipetest>

Change 136727 on 2003/12/10 by mmantor@FL_mmantorLT_r400_win

<added all vector ops>

Change 136641 on 2003/12/10 by llefebvr@llefebvre_laptop_r400_emu

Fix for vector TRUNC +/- 0 clamping issues.

Change 136555 on 2003/12/09 by mmantor@FL_mmantorLT_r400_win

add VectPipeTest that test the vector pipe of the sp.

Change 136540 on 2003/12/09 by llefebvr@llefebvre_laptop_r400_emu

This should fix TRUNC nan failures and DOT2Add INF and NAN failures.

Change 136519 on 2003/12/09 by llefebvr@llefebvre_laptop_r400_emu

Another pass on the SP files to be more HW lookalike.

Change 136387 on 2003/12/09 by llefebvr@llefebvre_laptop_r400_emu

This is the new SP files more HW lookalike. Also fixes DOT2ADD SRCC swizzle issues.

Change 136287 on 2003/12/08 by llefebvr@llefebvre_laptop_r400_emu

The TRUNC was rounding instead of clamping when dealing with negative numbers (LSB of mantissa only)

Change 135538 on 2003/12/04 by vromaker@vromaker_emul_r400_linux_marlboro

- updated vtx and pix thread buffer sizes

Change 135361 on 2003/12/04 by llefebvr@llefebvr_r400_emu_montreal

This should fix +/0 problems in CUBE (501-502). And TRUNC to +0 issue (trunc_02).

Change 134820 on 2003/12/02 by llefebvr@llefebvr_r400_emu_montreal

Fixing the CUBE opcode. It was reading from SRCB.X while if should have read from SRCA.W.

Change 134536 on 2003/12/01 by llefebvr@llefebvr_r400_emu_montreal

Fixing bad swizzle in the case of a co-issued 2 operand scalar op along with a 3 operand vector op. Emulator was doing x and w while it should have been z and w.

Change 133958 on 2003/11/25 by llefebvr@llefebvr_r400_emu_montreal

There was a parenthesis problem in the if causing the fix for the +/- zero not to work on the Muladd. This is fixing it.

Change 133810 on 2003/11/25 by llefebvr@llefebvr_r400_emu_montreal

Fixing Add/MulAdd +/-0 problem in the emulator

Change 133619 on 2003/11/24 by llefebvr@llefebvr_r400_emu_montreal

Fixing additionnal control flow entry in the SQ dumps.

Change 133415 on 2003/11/21 by llefebvr@llefebvr_r400_emu_montreal

Fixed the wrong function for the +/- Zero bug on the Muladd engine.

Change 133358 on 2003/11/21 by llefebvr@llefebvr_r400_emu_montreal

Adding RETAIN_PREV opcode to the scalar engine as opcode 0x32 (last one).
Opcode has syntax:

RETAIN_PREV Rx; Where Rx is any register. To just retain the predicate vector use:
RETAIN_PREV Rx@;

Fixed bad SRC selection for Tc in cube opcode.
Fixed +/- 0 problem in the multiply engine.

Change 133025 on 2003/11/20 by mearl@mearl_xenos_linux_orl

Fixed 2 primitive per clock interpolation bug in the emulator.

Change 132250 on 2003/11/17 by llefebvr@llefebvr_r400_linux_marlboro

Changing emulator and tests to meet with the new cube swizzles wich now are for SRCA zzxy (instead of zzyx). Also change the assembler to accept the new swizzle code.

Change 131172 on 2003/11/10 by llefebvr@llefebvre_laptop_r400_emu

This fixes KILL* and PRED_SET_INV opcodes when working with denorms.

Change 130817 on 2003/11/07 by llefebvr@llefebvr_r400_emu_montreal

Fixing SUB NaN sign propagation.
Fixing PRED* flush of denorms to 0.
Also fixing +/-0 flush to zero problems in multiple other opcodes.

Change 130473 on 2003/11/06 by llefebvr@llefebvr_r400_emu_montreal

Now using SRCB.z for input argument of the Z channel ma computation when Z wins in a cube opcode.

Change 130441 on 2003/11/06 by llefebvr@llefebvr_r400_emu_montreal

Removal of duplicate code.

Change 130357 on 2003/11/05 by llefebvr@llefebvr_r400_emu_montreal

Fixing the CUBE opcode. Was broken because the emulator wasn't taking into account that SRCA and B had a different set fo swizzles.

Change 130281 on 2003/11/05 by llefebvr@llefebvr_r400_emu_montreal

Fixing SRCC valid GPR valid channel.
Putting the SERIAL on the right line the the cubic pixel shader program.

Change 130216 on 2003/11/04 by mmantor@FL_mmantorLT_r400_win

<changes to enable standalone vector pipe random testbench>

Change 130154 on 2003/11/04 by llefebvr@llefebvr_r400_emu_montreal

Updated DOT2ADD opcode to propagate NAN from SRC C as well.

Change 130063 on 2003/11/04 by llefebvr@llefebvr_r400_emu_montreal

This should fix NAN sign propagation on the SUBs* opcodes (including SUB_PREV).

Change 129772 on 2003/11/03 by llefebvr@llefebvr_r400_emu_montreal

This fixes a problem were the dummy line in the SP->SX dump to signify a dump free done on memory export was places between phases 2 and 3 of an export instead of being between two exports. This change will only affect the TB_SQSPSX and TB_SX

Change 129642 on 2003/10/31 by llefebvr@llefebvr_r400_emu_montreal

New ExecuteVPInst function more flexible and hence more suitable to the testbench.

Change 129587 on 2003/10/31 by llefebvr@llefebvr_r400_emu_montreal

Instanciation of multiple memories in the emulator to allow for use of SQ_DEBUG_MISC.DB_EN_MEMORY_X and DB_READ_MEMORY register fields.

Change 129454 on 2003/10/30 by llefebvr@llefebvr_r400_emu_montreal

This should make the CUBE opcode match exactly HW even if SRCA and B are different.

Change 129388 on 2003/10/30 by llefebvr@llefebvr_r400_emu_montreal

The emulator does not clamp the W channel anymore on a clamped CUBE operation. This is in order to mimmic HW behavior.

Change 129348 on 2003/10/30 by mearl@mearl_xenos_linux_orl

Added two primitive interpolation back in.

Change 129150 on 2003/10/29 by llefebvr@llefebvr_r400_linux_marlboro

Increasing VC mini count to l1_fifo_size +2.

Change 129117 on 2003/10/28 by danh@danh_xenos_linux_orl

Changed sq_sx_pcaddr.dmp generation.

Change 128927 on 2003/10/28 by llefebvr@llefebvr_r400_emu_montreal

Fixing precision problem in all PRED* vector opcode. Emulator was rounding.

Change 128679 on 2003/10/27 by llefebvr@llefebvr_r400_emu_montreal

This fixes an emulator dump problem related to VSISR address when vtx_count is on but VSR_continued if off.

Change 128526 on 2003/10/24 by mearl@mearl_xenos_linux_orl

Took out two prim per clock to get regression to pass.

Change 128365 on 2003/10/24 by mearl@mearl_xenos_linux_orl

Added 2 primitive interpolation in SQ and SPI. Fixed a bug in sx_parameter_cache. Fixed synthesis
bugs in SC.

Change 128179 on 2003/10/23 by llefebvr@llefebvr_r400_emu_montreal

This should fix the bad data type on a position free done in the sp->sx emulator dump file. This will only affect the TB_SQSP,TB_SQSPSX and TB_SX test benches as they are the only ones using this field.

Change 127742 on 2003/10/22 by llefebvr@llefebvr_r400_linux_marlboro

Removed the warnings from the sp->sx trackers and sx->sp.
Now emulator is always executing the scalar instruction even in the case of a 3 operand vector opcode. This is to match with random shaders.

Change 127044 on 2003/10/16 by llefebvr@llefebvre_laptop_r400_emu

There was a bug in the post steer valid bits sent to the TP and VC. This is fixing it.

Change 127009 on 2003/10/16 by llefebvr@llefebvre_laptop_r400_emu

This fixed the pixel auto counter bug in the interpolators.

Change 126957 on 2003/10/16 by llefebvr@llefebvre_laptop_r400_emu

Now pushing the PERFORMANCE_STOP events to the RS.
Fixed the predicate jump bug (wasn't reading the correct bit in the control flow instruction).

Change 126833 on 2003/10/15 by dclifton@dclifton_xenos_linux_orl

Small change to log result for numbers less than 1.0 to match HW.

Change 126691 on 2003/10/15 by dclifton@dclifton_xenos_linux_orl

Another timing related change. Changed twos comp to ones comp on log
post-process (effects log of number less than 1.0). Aligned inputs
to high precision pipeline to reduce muxing. Improved carrysave add
of multiplier results. Regenerated math tables to reclaim precision
and fix roll-over mismatches.

Change 126641 on 2003/10/14 by llefebvr@llefebvre_laptop_r400_emu

Adding conditional_predicate_jmp and conditionnal_predicate_call. These are performance optimizations only and are not in use at the moment.

Change 126483 on 2003/10/13 by mearl@mearl_xenos_linux_orl

Fix One Prim Per Clock bug in sq_ptr_buff. Revert changes in sq_pix_ctl to make 2 prim interp changes easier. Put known primdata data on all quads across packer to iterator interface. Fix dumps for no_inc_pix_cnt signal.

Change 126362 on 2003/10/13 by rramsey@rramsey_xenos_linux_orl

Change sq_sp_interp dump so it contains all of the pass_count and wrap passes through the interpolator
Add spi_sp tracker (enabled with ENABLE_SPI_TRACKER define)

Change 125881 on 2003/10/09 by llefebvr@llefebvr_r400_emu_montreal

Now doing a MAX on the vector and scalar MOVA opcodes to match HW.

Change 125617 on 2003/10/08 by llefebvr@llefebvr_r400_emu_montreal

Now passing the ROM data at the end of each line in the dump file. This is for RSP usage in testbenches.

Change 125370 on 2003/10/07 by mearl@mearl_xenos_linux_orl

Fixed the SQ bug when bad pipe exists before a good pipe. Also, updated the RT trackers in the SC testbench.

Change 125266 on 2003/10/07 by llefebvr@llefebvr_r400_emu_montreal

Added RSP fields to the dumps to allow for easier testbench integration.

Change 125023 on 2003/10/06 by llefebvr@llefebvr_r400_emu_montreal

Cleaned up the COND_PRED_EXECUTE and PRED_LOOP code in the emulator. No bug fixes here, just made the code easier to read and removed dead paths.

Change 124883 on 2003/10/03 by llefebvr@llefebvr_r400_emu_montreal

Fixing issue with NANs and TRUNC,FLOOR,FRACT. Also, now SIN returns +/- 0 on input +/- 0.

Change 124851 on 2003/10/03 by llefebvr@llefebvr_r400_emu_montreal

Interpolation precision change to meet HW and timing.

Change 124843 on 2003/10/03 by llefebvr@llefebvr_r400_linux_marlboro

These are register changes related to new arbitration policies for the ALU engine.

Change 124643 on 2003/10/02 by llefebvr@llefebvr_r400_emu_montreal

Fixing hang on pred_optimization test.

Change 124518 on 2003/10/02 by llefebvr@llefebvr_r400_emu_montreal

This implements the cond_pred optimization in the emulator in order to match HW in the number of clause boundaries.

Change 124434 on 2003/10/01 by mmang@mmang_xenos_linux_orl

1. Turned on 3 simds in emulator (sc_interp.cpp, sq_block_model.cpp, and user_block_model.cpp).
2. Turned on 3 simds in rtl (sc_packer.v, tb_sqsp.v, and vgt.v).
3. Fixed bug in chip_vc.tree to get SQ_VC_simd_id and TC_VC_simd hooked up correctly.
4. Fixed bug in sc_packer.v related to having a 2 bit simd_id_sel.

Change 124011 on 2003/09/30 by llefebvr@llefebvr_r400_emu_montreal

Autoreg change to remove the _no_stalls opcodes and replace them with _pred_clean. Documentation only should not be a functionnal change. I had to remove some refrences to _no_stall in the emulator code as well.

Change 123879 on 2003/09/29 by llefebvr@llefebvr_r400_emu_montreal

Fixing some RSP problems on the Sq->TP/VC interfaces. These problems only show up when RSP is turned on (it is currently off).

Change 123610 on 2003/09/26 by llefebvr@llefebvr_r400_emu_montreal

Trying to FIX bad NAN handling on the scalar pipe.

Change 123497 on 2003/09/26 by llefebvr@llefebvr_r400_emu_montreal

Added the SP->SX RSP interface. Also added post_steered bits to all interfaces for tracker purposes only (not HW). These are the valid bits of the SPs AFTER the data was steered to go to the RSP.

Change 123278 on 2003/09/25 by llefebvr@llefebvr_r400_emu_montreal

Added TP redundant pipe. Also renamed all redundant pipes to use SP instead of SQ.

Change 123130 on 2003/09/24 by llefebvr@llefebvr_r400_emu_montreal

First pass at redundant pipe in the emulator. For now, only on the VC->SQ and SQ->VC interfaces.

Change 123082 on 2003/09/24 by mearl@mearl_crayola_linux_orl

tb files updated for ONE_PRIM_PER_CLOCK, bug fix in interpolators for ONE_PRIM_PER_CLOCK

Change 122815 on 2003/09/23 by llefebvr@llefebvr_r400_linux_marlboro

Added more perf counters to the SQ to account for SIMD2-3.

Change 122728 on 2003/09/23 by llefebvr@llefebvr_r400_emu_montreal

Made the autocount write to all channels like the HW does. Auto count in X, 0 in all other channels.

Change 122683 on 2003/09/23 by mearl@mearl_crayola_linux_orl

One primitieve per clock changes in the back of the SC and front of the SQ. Right now, the ONE_PRIM_PER_CLOCK define in header.v and SC_SQ_interface.v are needed for this change. Will update this to ONEPPC, since this already exists in header.v. Also, the sim.cfg file does not have an ifdef, so is hardcoded to one prim per clock.

Change 122677 on 2003/09/23 by dclifton@dclifton_xenos_linux_orl

Changed initial twos comp on exp opcode to ones comp for timing improvement in hardware.

Change 121928 on 2003/09/17 by llefebvr@llefebvre_laptop_r400_emu

Fixing multiple SIMD related stuff.

Change 121897 on 2003/09/17 by dclifton@dclifton_crayola_linux_orl

Change in HW structure to improve timing

Change 121632 on 2003/09/16 by llefebvr@llefebvre_laptop_r400_emu

Made the XYs be sent as floating point numbers instead of fix point so you can use them directly in the shader. Also modified regress_e tests that this change broke. Also added register fields for SIMD memory control.

Change 120731 on 2003/09/11 by llefebvr@llefebvr_r400_emu_montreal

I have removed the predication optimization on waterfall from the emulator. The HW does not do it. Lines are marked with @@ OPTIMIZE so we can know what was removed in order to put it back if needs be.

Change 120546 on 2003/09/10 by llefebvr@llefebvr_r400_emu_montreal

Adding the scalar engine change. This change is currently disabled. To enable, change line # 105 of sclarfunc.h from:

```
__int64 ComputeMantissa6X(unsigned int point, unsigned int slope,
                          unsigned int slope_derivative,
                          unsigned int sloped_derivative,
                          unsigned int delta, unsigned int ddelta,
                          unsigned int dddelta, MeParam2 *params,
                          int gen_type, int new_struct = 0);
```

to:

```
__int64 ComputeMantissa6X(unsigned int point, unsigned int slope,
                          unsigned int slope_derivative,
                          unsigned int sloped_derivative,
                          unsigned int delta, unsigned int ddelta,
                          unsigned int dddelta, MeParam2 *params,
                          int gen_type, int new_struct = 1);
```

Change 120264 on 2003/09/09 by llefebvr@llefebvr_r400_emu_montreal

Now writing the channel valid mask in hex in the dumps.

Change 120058 on 2003/09/08 by chammer@chammer_r400_win

Changed width of dealloc from 3 to 4 bits since it can now accumulate to 8 in 1 prim per clock mode.
Load prim data into 'bad pipes' to match hardware.

Change 119590 on 2003/09/05 by llefebvr@llefebvre_laptop_r400_emu

Rearranged the functions in the shader pipe for SP testbench purposes. No functionnal changes per say.

Change 118543 on 2003/08/28 by llefebvr@llefebvr_r400_emu_montreal

Adding thread type to SX interfaces for memory export validation.
Also made sure to invalidate the Parameter cache interface reads whenever generating a parameter.

Change 118385 on 2003/08/27 by llefebvr@llefebvr_r400_emu_montreal

Added the thread Id identifier to interfaces for tracking purposes of memory export tests.

Change 118345 on 2003/08/27 by llefebvr@llefebvr_r400_emu_montreal

Missing a file in the changelist.

Change 118332 on 2003/08/27 by llefebvr@llefebvr_r400_emu_montreal

Massive changelist of mostly non-functionnal stuff with two exeptions:

1) Now checking for INF in the DOT path and doing as per the numerics spec.
2) Changed the VC mini instruction count limit from 16 to 32 as instructed by Brian Buchner.

The rest of the changelist is the first pass to correct the invalid channels in the GPRs in order for them not to be compared. This is mostly only adding bits at the end of the dumps for the trackers to have the information needed to know what's valid and what's not. Interfaces changed are:

1) Internal SP GPR
2) SP->SX
3) SX->SP
4) SX->PA
5) SX->RB
6) SX->Interp (PC data)
7) SP interpolators

Change 118211 on 2003/08/26 by rramsey@rramsey_crayola_linux_orl

changes to get fake_last set correctly in tp_sqsp.dmp

Change 117786 on 2003/08/22 by llefebvr@llefebvre_laptop_r400_emu

The loop index clamping wasn't done correctly as I was trying to force negative values on unsigned variables.

Change 117590 on 2003/08/21 by llefebvr@llefebvre_laptop_r400_emu

I broke pretty much all waterfall tests with my previous checkin. This is a fix.

Change 117545 on 2003/08/21 by llefebvr@llefebvre_laptop_r400_emu

This is an attempt to fix the bug 2515. I made sure a pixel wasn't considered in subsequent passes if it was written. I will need sunshine to test it as I can't run the app.

Change 117283 on 2003/08/20 by llefebvr@llefebvre_laptop_r400_emu

For parameter cache (interpolators) exports, I was using a different function to output the results to a file. I did not bother opening up the file in this function as I suspected the position export function would open it for me. Obviously, if you export position last this doesn't work. I

modified the function to check for file open and open it if not. This fixes the numerous bugs the compiler guys were seing when exporting position last. This bug would only show up if dump files are activated.

Change 117162 on 2003/08/19 by llefebvr@llefebvre_laptop_r400_emu

Fixing CUBE instruction by move the *2.0 from red to blue.

Change 117149 on 2003/08/19 by llefebvr@llefebvre_laptop_r400_emu

Now clamping to -256 if overflow of the loop index register.

Change 116305 on 2003/08/13 by chammer@chammer_r400_win

Added one prim per clock to SC. This code is ifdef'd and can be enabled by uncommenting out //#define ONE_PRIM_PER_CLK in gfx/sc/sc_types.h as well as in gfx/sq/sc_sq.h

Change 115549 on 2003/08/08 by llefebvr@llefebvr_r400_emu_montreal

Now writing DEADDEAD in the parameter caches at alloc time.
Any DEADDEAD pixel or vertex will go thru CLAMP unchange is it is a DEADDEAD.

Change 115328 on 2003/08/07 by llefebvr@llefebvr_r400_emu_montreal

The HW has a 26 bits normilizer so I increased the emulator's precision to match. This fixes r400sq_ripple_01.cpp. I had to re-goldenize 1 test in the regress_e suite.

Change 115032 on 2003/08/05 by grayc@grayc_crayola2_linux_orl

added back Laurent changes for sx performance counters
modified sx.v for new performance register names

Change 114997 on 2003/08/05 by llefebvr@llefebvr_r400_emu_montreal

Forgot to flush denorms prior to check for == 0. This was causing some tests to missmatch on the SX->SP parameter sub interface.

Change 114116 on 2003/07/31 by grayc@grayc_crayola2_linux_orl

backing out changes for SX Perf Counters

Change 114066 on 2003/07/30 by llefebvr@llefebvr_r400_emu_montreal

Aligned correctly the sq_sp_interp.dmp
Made a small change to generate -0 in the param_bub engine of the SX when both operands are the same and SRCA is negative. This is to match the HW algorithm.

Change 114014 on 2003/07/30 by llefebvr@llefebvr_r400_emu_montreal

Made the SX performance counters unified in one SX. Also added some performance counters to monitor the SX->RB performance. All of these are non-windowed yet.

Change 113865 on 2003/07/29 by llefebvr@llefebvr_r400_emu_montreal

This toggles the emulator into a single/multiple SIMD engine. Can toggle on/off by undef/def line 41 of sq_block_model.cpp (#define ENABLE_MULTIPLE_SIMDS). Currently OFF.

Change 113818 on 2003/07/29 by llefebvr@llefebvr_r400_emu_montreal

Now routing SIMD id to the SQ and TP dumps.
Added 4 GPR banks to support up to 4 SIMD engines all having the same GPR address but different data.
Added 4 allocation engines to generate these addresses.
The number of SIMD engines is user defined it is a #define in sq/user_block_model.h, current setting is 4 but only 2 are used as the SC and VGT never send more than that for now.

Change 113562 on 2003/07/28 by llefebvr@llefebvr_r400_emu_montreal

Cleaned and Fixed memory exports.
Made the resource register pick the VC if a vertex thread is entered.

Change 112593 on 2003/07/23 by llefebvr@llefebvre_laptop_r400_emu

Fixing infinite loop problem on 0 count exec_ends because of not setting correctly the end of program flag in the SQ.

Change 112528 on 2003/07/22 by hartogs@fl_hartogs2

Made the simd_id field between the VGT and the SQ two bits for up change to three or more SIMD sets.

Change 112332 on 2003/07/22 by llefebvr@llefebvre_laptop_r400_emu

I had the old output order for cube: ma,faceid,sc,tc. I changed it for the new one: tc,sc,ma,faceid.

Change 112254 on 2003/07/21 by llefebvr@llefebvre_laptop_r400_emu

This is the CUBE opcode change that takes into account the recent HW change. I also modified one test case that was wrongfully picking W as the FACEID (it is Y).

Change 111835 on 2003/07/18 by llefebvr@llefebvr_r400_emu_montreal

Splitting the ALU CONSTANT perf counters into SIMD0 and 1. I have added the SIMD1 version of the counters at the end of the enumeration not to disturb again the current order.

Change 111444 on 2003/07/16 by llefebvr@llefebvr_r400_emu_montreal

Added sin/cos corner cases.

Change 111399 on 2003/07/16 by llefebvr@llefebvr_r400_emu_montreal

Fixing HOS anomaly because of bad SIMD setting in the SQ.
Adding the auto-generated parameter in the assert for too many parameters in the RT SQ.

Change 111005 on 2003/07/14 by llefebvr@llefebvr_r400_emu_montreal

Changing the RT assert to <= instead of <

Change 111001 on 2003/07/14 by llefebvr@llefebvr_r400_emu_montreal

Adding the SIN/COS opcodes in the emulator.

Change 110861 on 2003/07/14 by llefebvr@llefebvr_r400_emu_montreal

SQ now only reading TP interface data when Phase == 3 AND TP_SQ_VALID. Also added an assert when trying to interpolate more than 4 parameters in RT mode.

Change 110814 on 2003/07/14 by jhoule@jhoule_r400_win_lt

Added separate valids for TP_SQ and TP_SP data.
This will eventually allow simpler HiColor data return from TP.

Change 110319 on 2003/07/10 by llefebvr@llefebvr_r400_emu_montreal

Wasn't setting the VC counters correctly in the SQ. Was causing a hang whenever an app would use a mini-fetch.

Change 109729 on 2003/07/08 by llefebvr@llefebvr_r400_emu_montreal

Fixed the pixel counter in the SQ. This value is not used yet.

Change 108547 on 2003/06/30 by lseiler@lseiler_r400_win_marlboro

Push out MemExports before processing certain events.

Change 108285 on 2003/06/27 by llefebvr@llefebvre_laptop_r400_emu

Swapping write priority to a GPR. Was vector has priority over scalar, now IS scalar has priority over vector.

Change 108154 on 2003/06/26 by hartogs@fl_hartogs2

Hopefully fixed VGT alloc/dealloc for multi-SIMD vector sets

Added simd_id fields to vgt_sq interface and vgt_ccgen interface
Put pipedisable comments into several dump files.
Put an "Assert(0)" into the sq_block_model.cpp to prevent access violation.

Change 107916 on 2003/06/25 by llefebvr@llefebvre_laptop_r400_emu

Made all the necessary changes for the color buffers to be enlarged without actually enlarging them so I don't break anything.

Change 107624 on 2003/06/24 by llefebvr@llefebvre_laptop_r400_emu

Now enlarging valid bit mask to be quad aligned for pixel vectors (in order to get LOD right to the TP).

Change 107575 on 2003/06/24 by llefebvr@llefebvre_laptop_r400_emu

Added back pressure ability from the VC to the SQ using the Mega and mini counters.

Change 107515 on 2003/06/23 by mzini@mzini_r400_win

Added vc_sq_sync interface

Change 107512 on 2003/06/23 by mzini@mzini_r400_win

Created VC_SQ_Sync interface

Change 107026 on 2003/06/19 by llefebvr@llefebvr_r400_linux_marlboro

1) Added a guard bit to the parameter sub engine of the SX in both the emulator and HW this was causing a failure on a WQL test.
2) Fixed zero detection problem in parameter_sub engine of the HW were an explicit 1 was added all the time even when the number was 0.0. This was causing r400sx_wrapper_01.cpp to fail (this is a test that I wrote to duplicate the WQL test that was failing in order to run it on HW).

Change 106833 on 2003/06/18 by llefebvr@llefebvr_r400_emu_montreal

Added loop index clamping to range -256,255

Change 106809 on 2003/06/18 by llefebvr@llefebvr_r400_emu_montreal

Was using the textureCF machine for the VC incorrectly. Caused the VC to fail.

Change 106526 on 2003/06/17 by llefebvr@llefebvr_r400_emu_montreal

Was incorectly reading unitialized data.

Change 106470 on 2003/06/17 by rramsey@rramsey_crayola_linux_orl

Change interpolators so sx_sp_pcdata dump is not written
for param_gen or gen_index passes (param cache is not actually
read for those)
Clean up sqsp_interp dump

Change 106404 on 2003/06/16 by llefebvr@llefebvr_r400_emu_montreal

Initializing GPRS with DEADDEAD for ferret tracking purposes. Each thread is initialized with this at GPR allocation time.

Change 106347 on 2003/06/16 by llefebvr@llefebvr_r400_emu_montreal

Made the wrapping test >= instead of >

Change 106339 on 2003/06/16 by llefebvr@llefebvr_r400_emu_montreal

Fixing SQ->VC bad header in dump.
Fixing bad mova clamping.

Change 106263 on 2003/06/16 by mzini@mzini_r400_win

Removed y and z from the index field in sq_vc.dmp file AGAIN!

Change 106093 on 2003/06/13 by llefebvr@llefebvr_r400_emu_montreal

Emulator now clamps the address register to the range -256...255.

Change 106000 on 2003/06/13 by llefebvr@llefebvr_r400_emu_montreal

Fixing SQ->VC dump to be correctly aligned.

Change 105865 on 2003/06/12 by llefebvr@llefebvr_r400_emu_montreal

Fixing bad write mask setup on waterfall of lots of vertexes. Also enable waterfall_optimize on exports.

Change 105795 on 2003/06/12 by llefebvr@llefebvr_r400_emu_montreal

Fixing waterfalling on a MOVA instruction.

Change 105772 on 2003/06/12 by llefebvr@llefebvr_r400_emu_montreal

Fixed a bug that caused generation of INF in the interpolators when param 0 was INF regardless of param 1 and 2. Also realigned the sq_sp_interp.dmp.

Change 105645 on 2003/06/11 by llefebvr@llefebvr_r400_emu_montreal

There was a problem in the way the masks were set when waterfalling. If multiple vertexes were using the same constant on the second pass, only one of them was set.

Change 105336 on 2003/06/10 by llefebvr@llefebvr_r400_emu_montreal

Waterfalling wasn't tied to write mask of GPRs hence creating corruption when running in optimized mode.

Change 105295 on 2003/06/10 by llefebvr@llefebvr_r400_emu_montreal

Bad initialization for optimized waterfalling that caused linux to loop indefinitelly.

Change 105276 on 2003/06/10 by llefebvr@llefebvr_r400_emu_montreal

Changing the order of waterfalling to match HW on the interface. Emulator was doing LSB->MSB while HW was doing MSB->LSB.

Change 105001 on 2003/06/09 by llefebvr@llefebvr_r400_emu_montreal

Added waterfall write masks to the emulator.
Fixed the ALU instruction dump file to work with waterfall.

Change 104944 on 2003/06/09 by llefebvr@llefebvr_r400_emu_montreal

Modified the emulator to generate the same number of passes than the HW on a waterfall instruction. Only the number of passes is correct at this point not the write mask.

Also corrected 2 MOVA tests where the address register wasn't loaded correctly before use.

Change 104090 on 2003/06/04 by llefebvr@llefebvr_r400_emu_montreal

Made 3 different counters for fetch, cst and instructions.

Change 104059 on 2003/06/04 by llefebvr@llefebvr_r400_emu_montreal

Added a RT only instruction shader dump.

Change 103964 on 2003/06/04 by llefebvr@llefebvr_r400_emu_montreal

1) Adding normalization stage to the SX
2) Fixing unititialized loop variables in the SQ.
3) Fixing RT/Normal constant and instruction loads (arbitrated on a 32 bit basis now)
4) Fixing cylindrical wrap problem in SX
5) Added performance counters for SIMD0/1

Change 103846 on 2003/06/03 by rramsey@rramsey_crayola_linux_orl

Fix a problem with the control flow dump logic that was causing
zero-count execs to be left out of the file

Change 103806 on 2003/06/03 by rramsey@RRAMSEY_P4_r400_win

fix comment line in SqSpInterp_Dump

Change 103675 on 2003/06/02 by mzini@mzini_r400_win

Added simd_id to sq_vc and vc_sq dumps

Change 103552 on 2003/06/02 by llefebvr@llefebvr_r400_emu_montreal

Added SIMD ID to TP interfaces.

Change 103230 on 2003/05/30 by mzini@mzini_r400_win

Added simd_id to SQ_VC and VC_SQ

Change 101734 on 2003/05/20 by llefebvr@llefebvre_laptop_r400_emu

fixing scalar trunc problem.

Change 100741 on 2003/05/13 by llefebvr@llefebvr_r400_emu_montreal

Fixing range undeflow check of GPRs when using loop indexing and negative step.

Change 100446 on 2003/05/12 by llefebvr@llefebvr_r400_emu_montreal

was not checking for < BASE for constant indexing range checking.

Change 100178 on 2003/05/09 by llefebvr@llefebvr_r400_emu_montreal

bug with the VC arbiter.

Change 99938 on 2003/05/08 by hwise@fl_hwise_r400_win

SC, SQ, & VGT Emulator Update:
- Added shader pipe disable support for multiple SIMDs

Change 99865 on 2003/05/08 by llefebvr@llefebvr_r400_emu_montreal

Fog changes: Serialized fog in the SP after the default settings for exports.
Fixed assembler to put 0 in scalar write mask for BW compatibility with od shaders.

Change 99556 on 2003/05/07 by llefebvr@llefebvr_r400_emu_montreal

fixing the RT assertion.

Change 99443 on 2003/05/06 by llefebvr@llefebvr_r400_emu_montreal

Added an assert for bad constant registry setting regarding RT.

Change 99140 on 2003/05/05 by lseiler@lseiler_r400_win_marlboro

Eliminate two vcc warning messages

Change 98355 on 2003/04/30 by hwise@fl_hwise_r400_win

ROM/GFX Emulator Update:
1) Removed register fields from ROM_BAD_PIPE_DISABLE_REGISTER
- DISABLE_SP_VTX
- DISABLE_SP_PIX
2) Added register ROM_SIMD_PIPE_DISABLE_REGISTER with fields
+ DISABLE_SIMD0_VTX
+ DISABLE_SIMD0_PIX
+ DISABLE_SIMD1_VTX
+ DISABLE_SIMD1_PIX
+ DISABLE_SIMD2_VTX (reserved for future use)
+ DISABLE_SIMD2_PIX (reserved for future use)
+ DISABLE_SIMD3_VTX (reserved for future use)
+ DISABLE_SIMD3_PIX (reserved for future use)
3) Fixed bug in ROM block where post write trigger function for
ROM_BAD_PIPE_DISABLE_REGISTER was not named correctly in
rom.mblk files causing function to never be called
4) Added post write trigger function for new register
ROM_SIMD_PIPE_DISABLE_REGISTER
5) Updated PA, SC, SQ, and VGT blocks to use new SIMD0 fields
when determining bad and/or disabled pipes
6) Added "TO DO" comment where SIMD1 logic needs to be added
7) Modified InitRomStraps() to use ROM.ROM_BAD_PIPE_FUSE_REG.devForce()
to set the LASER_FUSES rather than ROM.ROM_BAD_PIPE_FUSE_REG.write()
since this is a read-only register without a default setting

Change 98272 on 2003/04/30 by llefebvr@llefebvr_r400_emu_montreal

This should fix Ken's bug. Ken please confirm by running this in your sand box...

Change 97570 on 2003/04/25 by llefebvr@llefebvre_laptop_r400_emu

Added SIMD1_DISABLE register field.
Added MEMORY_READ register field.
Added perf control to control performance dumps.
Fixed the overwrite I made in cf_machine.cpp with previous checkin.

Change 97404 on 2003/04/24 by llefebvr@llefebvre_laptop_r400_emu

Making the SX more HW alike for memory exports.
Fixing the wrapping bug in the control flow machine (related to jumping in the instruction store).

Change 97321 on 2003/04/24 by mmantor@FL_mmantorLT_r400_win

<fixed a bug with sx_sp_pcdata.dmp. The emulator was making the incorrect number of passes to do interpolation>

Change 97064 on 2003/04/23 by llefebvr@llefebvre_laptop_r400_emu

Added NO_SERIAL register.
Changed pred_set opcodes to use W instead of X (may break some SQ tests that will have to be fixed).

Change 97003 on 2003/04/22 by llefebvr@llefebvre_laptop_r400_emu

Added end of shader markers to shaders dump.

Change 96907 on 2003/04/22 by llefebvr@llefebvre_laptop_r400_emu

Added comments to SX dumps.
Added VC dumps.
Made emulator always stop on serial so it is deterministic.

Change 96369 on 2003/04/18 by llefebvr@llefebvr_r400_emu_montreal

adding some more code t the VC.
Fixed thread going always back on serial.

Change 96001 on 2003/04/16 by frising@frising_r400_win_marlboro

Move computation of a_mul_b_exp_flip_predict before a_mul_b_exp in muladd. This was the original intention and I suspect a copy/paste error at some point. In any case, it should not change the results. I checked with AndyG and apparently the HW is already doing this.

Change 95739 on 2003/04/15 by llefebvr@llefebvr_r400_emu_montreal

added the new performance counters in the SQ.

Change 95669 on 2003/04/15 by llefebvr@llefebvr_r400_emu_montreal

New dumps with shaders and number of vertexes + pixels per shader. Neede to validate load on the R500.

Change 95601 on 2003/04/15 by llefebvr@llefebvr_r400_emu_montreal

The SQ now sends data to the VC. A dump: sq_vc.dmp has also been added for test bench purposes.

Change 95456 on 2003/04/14 by llefebvr@llefebvr_r400_emu_montreal

Added the SQ->VC interface and VC->SQ interface.

Change 95216 on 2003/04/11 by llefebvr@llefebvr_r400_emu_montreal

Added a VC empty shell. Need to add the interface to it. SQ is ready.

Change 94942 on 2003/04/10 by llefebvr@llefebvr_r400_emu_montreal

changed the comment name for free_done_position to FDP

Change 94940 on 2003/04/10 by llefebvr@llefebvr_r400_emu_montreal

Adding pos_free_done to the sp_sx.dmp for the SX testbench.

Change 94875 on 2003/04/10 by llefebvr@llefebvr_r400_emu_montreal

Sx now sends R400_NANs to the PA when VS_EXPORT_MODE == 7.
Added number of scalar operands to SP_in_dump.

Change 94729 on 2003/04/09 by llefebvr@llefebvr_r400_emu_montreal

Added the bit necessary to add the VC engine. Also modified the control flow instruction.

Change 94688 on 2003/04/09 by grayc@grayc_crayola2_linux_orl

correct filename

Change 94344 on 2003/04/07 by grayc@grayc_crayola2_linux_orl

move appending of $TestPath into dump_point class for the rb,sx,sq,sc blocks

Change 94334 on 2003/04/07 by llefebvr@llefebvr_r400_emu_montreal

Fixing small discrepencies in the SP dumps. I also had a bad addressing on constants when using constant 0 on SRC B. Please Gang initiate a driver emulator drop and confirm this is fixed indeed.

Change 93911 on 2003/04/04 by llefebvr@llefebvr_r400_emu_montreal

I was not pushing the IP to and from the stack in the right order whenever moving to and from the RS. Also, now using a single table line in the SX export table when doing mem-exports and finxing additional entries in CF dump.

Change 93644 on 2003/04/03 by llefebvr@llefebvr_r400_emu_montreal

added channel masking to the SX.

Change 93506 on 2003/04/02 by llefebvr@llefebvr_r400_emu_montreal

adding jump/call address to the dump.

Change 93480 on 2003/04/02 by llefebvr@llefebvr_r400_emu_montreal

There was an error in the dump comment line for number of bits in EXEC subword.

Change 93403 on 2003/04/02 by llefebvr@llefebvr_r400_emu_montreal

Adding CONTEXT_DONE events to the RS (not sending them to the CP).
Fixing the problem with the SP->SX interface (was making 1 too many transfert because of the dummy line inserted in the dump).

Change 92966 on 2003/03/31 by llefebvr@llefebvr_r400_emu_montreal

Added the dummy free done line in case there is no export as a last clause instruction.

Change 92927 on 2003/03/31 by llefebvr@llefebvr_r400_emu_montreal

removing the | from the CF,ALU and RS dumps, it confuses the PLI routines.

Change 92640 on 2003/03/28 by llefebvr@llefebvre_laptop_r400_emu

Added the instruction store read address to the CF instruction and ALU dumps.

Change 92630 on 2003/03/28 by llefebvr@llefebvre_laptop_r400_emu

Fixing MAX4 problem with NANs.

Change 92626 on 2003/03/28 by llefebvr@llefebvre_laptop_r400_emu

Fixing the write enables of the SP dumps.

Change 92563 on 2003/03/28 by llefebvr@llefebvre_laptop_r400_emu

Added number of operands in the SP dumps.

Change 92409 on 2003/03/27 by llefebvr@llefebvre_laptop_r400_emu

Fixing the sign extention.

Change 92390 on 2003/03/27 by llefebvr@llefebvre_laptop_r400_emu

fixing more alloc problems with events that were hanging SQ and SC tests.

Change 92227 on 2003/03/26 by llefebvr@llefebvre_laptop_r400_emu

This should fix that. Let me know.

Change 92184 on 2003/03/26 by llefebvr@llefebvre_laptop_r400_emu

The SQ was not allocating in order for colors. This was causing the SX to missbehave. This is fixed.

Change 91585 on 2003/03/21 by llefebvr@llefebvr_r400_emu_montreal

Fixing wrong aluId number in arbiter.
Fixing mem exports some more.

Change 91363 on 2003/03/20 by llefebvr@llefebvr_r400_emu_montreal

fliping mask bit order

Change 91261 on 2003/03/20 by llefebvr@llefebvr_r400_emu_montreal

Major changes to the SQ/SX. Modified the SX to accept out of order exports and made it closer to HW. Also now matching the HW interface for allocs/deallocs in the SX.

Change 91256 on 2003/03/20 by llefebvr@llefebvre_laptop_r400_emu

fixing predication bug

Change 91169 on 2003/03/20 by rramsey@RRAMSEY_P4_r400_win

fix typos in two dump file names (pix/vtx was swapped for control flow dumps)

Change 90027 on 2003/03/13 by llefebvr@llefebvr_r400_emu_montreal

Fixed the Nan Check.
Added more fields to sp_sx.dmp and sx_sq_addr.dmp for Sx testbench.

Change 89938 on 2003/03/13 by rramsey@RRAMSEY_P4_r400_win

swap order of d0,1,2 in dump header

Change 89768 on 2003/03/12 by llefebvr@llefebvr_r400_emu_montreal

changing name export to Export because of name clash on linux.

Change 89716 on 2003/03/12 by llefebvr@llefebvr_r400_emu_montreal

SP data dumps.

Change 89381 on 2003/03/10 by llefebvr@llefebvr_r400_emu_montreal

Added dumps for ALU instructions.

Change 89037 on 2003/03/07 by llefebvr@llefebvr_r400_emu_montreal

fixing dumps and memory export test

Change 89017 on 2003/03/07 by llefebvr@llefebvr_r400_emu_montreal

Modifiyng dumps not to exceed 32 bits/col (PLI need)

Change 89007 on 2003/03/07 by llefebvr@llefebvr_r400_emu_montreal

Added control flow instruction dumps.

Change 88898 on 2003/03/06 by rramsey@RRAMSEY_P4_r400_win

Add new block level dumps to SQ for the vertex input controller (enabled with SqDump >1)
sq_sp_visr_wr.dmp - writes to staging registers
sq_sp_vec_gpr.dmp - reads from staging regs/writes to gprs
sq_vec_gpr_req.dmp - gpr allocate requests from the vertex controller

Change 88859 on 2003/03/06 by llefebvr@llefebvr_r400_emu_montreal

fixing 2D bug due to GPR addressing change to match HW.

Change 88737 on 2003/03/06 by grayc@grayc_crayola_linux_orl

changes for dump files

Change 88687 on 2003/03/05 by llefebvr@llefebvr_r400_emu_montreal

Fixing +/-0 bug in emu and type setting.

Change 88554 on 2003/03/05 by llefebvr@llefebvr_r400_emu_montreal

More SQ RS dumps.

Change 88079 on 2003/03/03 by llefebvr@llefebvre_laptop_r400_emu

Changed vertex GPR addressing to match HW exactly.

Change 87922 on 2003/03/01 by mmantor@FL_mmantorLT_r400_win

1. fixed a texture wrap bug.
2. fixed an sc hang for large vertices with lots of primitives culled where more than the max parameter cache space (2.5 times) could be required to operate. Put a fix that forces a partial fill of any vector being assembled if the previous vector had more than 4 outstanding deallocates out and a new vector (fpos) arrives at the packer input. This forces all the de-allocates to happen on the partial fill vector which will free pc space and allow the final vertex vector to process and etc. Added outstanding deallocate count to dump files sc_sq and sc_pix_vect_grp_out

Change 87326 on 2003/02/27 by llefebvr@llefebvre_laptop_r400_emu

Added dumps for the reservation stations.

Change 87112 on 2003/02/26 by llefebvr@llefebvre_laptop_r400_emu

added predicate (export write mask) to the SP->SX interface. Also added them in emulator dump.

Change 87027 on 2003/02/26 by llefebvr@llefebvre_laptop_r400_emu

Fixed Trunc.

Change 86925 on 2003/02/25 by llefebvr@llefebvre_laptop_r400_emu

modified dealloc code slightly to improve debugging of deallocs.

Change 86890 on 2003/02/25 by llefebvr@llefebvre_laptop_r400_emu

Fixing Inf and DOTs.
Aligning RT_CONSTANTS with cleaner register boundaries.

Change 86634 on 2003/02/24 by mmantor@FL_mmantorLT_r400_win

added thread id to sp->sx interface dump
fixed sx to sp parameter data dump
added sq to sx pc_ptr dump
added texture cylinderical wrap control to sq

Change 86257 on 2003/02/22 by mmantor@FL_mmantorLT_r400_win

Set ClampedAddr = Addr before clamping so if no reason to clamp the address is defined. Stops emulator from crashing.

Change 86096 on 2003/02/21 by llefebvr@llefebvr_r400_emu_montreal

That should do it. I wasn't carefull enough about the clamping rules to the GPRs when the pointer was in fact used to read constants...

Change 85184 on 2003/02/19 by llefebvr@llefebvr_r400_emu_montreal

fixed clamping of -0 to +0.

Change 85039 on 2003/02/18 by llefebvr@llefebvr_r400_emu_montreal

Flushing denorms to 0 is now done on all three parameters (I forgot to do it on P0 in the emulator).

Change 84940 on 2003/02/18 by llefebvr@llefebvr_r400_emu_montreal

Fixing clamping problem in the emulator.

Change 84802 on 2003/02/17 by llefebvr@llefebvr_r400_emu_montreal

Code cleanup and fixing the problem with INF on the recip opcode.

Change 84554 on 2003/02/14 by llefebvr@llefebvr_r400_emu_montreal

fixing the clamp of the NANs to 0 and 1. They now go thru the clamp modifier unchanged.

Change 84487 on 2003/02/14 by llefebvr@llefebvr_r400_emu_montreal

repairing instruction count perf counters.

Change 84204 on 2003/02/13 by llefebvr@llefebvr_r400_emu_montreal

DOT now returns an R400_NAN instead of the NAN it was provided with.

Change 83625 on 2003/02/11 by llefebvr@llefebvr_r400_emu_montreal

there was a typo in the interface. The context id was sent as the channel mask and vice versa to the TP...
Also added PC exports to the sp_sx.dmp file for the SQ/SP testbench.

Change 83566 on 2003/02/11 by hartogs@fl_hartogs2

Modified dump sp_sx.dmp to be more friendly for testbench trackers.

Change 83491 on 2003/02/11 by llefebvr@llefebvr_r400_emu_montreal

adding first of new context arbitration restriction for position export.

Change 83290 on 2003/02/10 by llefebvr@llefebvr_r400_emu_montreal

changing default primitive type to be PRIM_NROMAL instead of RT.

Change 83285 on 2003/02/10 by mmantor@FL_mmantorLT_r400_win

fixed a bug induced by last change that caused a the program to crash when sq_dumps where not enabled.

Change 83166 on 2003/02/10 by mmantor@FL_mmantorLT_r400_win

1. added state data and missing terms to the sc_packer dump.
2. added a dump for the sx_sp interface that delievers attribute data to the interpolators. Reoganized the sq ProcessPixel and Interpolate code to enable these dumps. Still need to make futher modifications to make dump work properly for cylinderical wrap.

Change 83000 on 2003/02/07 by sallen@sallen_r400_lin_marlboro

ferret: tp_sq add ferret bits, to be unused soon
clean up a few comparitor outputs

Change 82812 on 2003/02/07 by llefebvr@llefebvr_r400_emu_montreal

Added a new performance select counter to count the numbers of stalls due uniquelly to position export buffer full in the SX.

Change 82646 on 2003/02/06 by llefebvr@llefebvr_r400_emu_montreal

now when overflow shifting we do this signed like the HW.

Change 82206 on 2003/02/05 by llefebvr@llefebvr_r400_emu_montreal

added channel mask to the SQ->TP interface for ferret interface checking.

Change 82048 on 2003/02/05 by llefebvr@llefebvr_r400_emu_montreal

Masking out bits [31:24] on the read of a CF_LOOP register. Now, the emulator will always return 00 for bits [31:24] when reading such a register.

Change 81837 on 2003/02/04 by llefebvr@llefebvr_r400_emu_montreal

Trying to fix cyl wrapping bug. Also removed some warnings in sq_alu.h.

Change 81790 on 2003/02/04 by llefebvr@llefebvr_r400_emu_montreal

New muladd and dot code as provided by Tom.

Change 81577 on 2003/02/03 by llefebvr@llefebvr_r400_emu_montreal

Emulator not behaving correctly when interpolating more than 1 param (in flat shading mode). This is fixed and I am writing directed tests to make sure HW is correct as well.

Change 81430 on 2003/02/03 by llefebvr@llefebvr_r400_emu_montreal

New muladd code with exponent prediction.

Change 81246 on 2003/02/01 by sallen@sallen_r400_lin_marlboro

ferret: comparitor work continues
make files now supported (start to phase out cons)
initial tp4 checking
add emu changes for mem id, etc

Change 81131 on 2003/01/31 by llefebvr@llefebvre_laptop_r400_emu

Added loop relative indexing modes to texture fetches.

Change 81096 on 2003/01/31 by llefebvr@llefebvre_laptop_r400_emu

The emulator was doing the AND of all channels in order to kill a pixel or not. The right thing to do is the OR as specified by the DX API.

Change 81054 on 2003/01/31 by llefebvr@llefebvre_laptop_r400_emu

Enabling GPR writes on the kill instructions (where previously not writting anything).

Change 81015 on 2003/01/31 by llefebvr@llefebvre_laptop_r400_emu

fixing a MOVA problem in the emulator when dealing with large primitives.

Change 80459 on 2003/01/29 by llefebvr@llefebvre_laptop_r400_emu

Fixing bug in SQ where SQ was reading instruction 0 when it wanted to do a NOP.
Fixing overflow in DOT introduced by previous change (revert).

Change 80400 on 2003/01/29 by lseiler@lseiler_r400_win_marlboro

fixed linux warnings

Change 80277 on 2003/01/29 by llefebvr@llefebvre_laptop_r400_emu

Refined the overflow check for the DOT opcodes add.

Change 80103 on 2003/01/28 by llefebvr@llefebvre_laptop_r400_emu

changing the order of the processing of the DOT product to match HW. This does not modify the output.

Change 79829 on 2003/01/27 by llefebvr@llefebvre_laptop_r400_emu

Fixing the access violation exeption.
Also making RealDOT the default setting for the emulator. Emulator should now be 100% HW accurate in the SP.

Change 79344 on 2003/01/24 by mmantor@fl_mmantorxp_r400_win

changed order of new_vector and deallocate_pc in the 1clk transfer. The sc_random test identified a bug where a new_vector and it's deallocate_pc can come in one transfer and the seq scheduled their execution incorrectly.

Change 78935 on 2003/01/23 by llefebvr@llefebvr_r400_emu_montreal

Now flushing denorms resulting of the normalization at the output of the interpolators.
Fix for the depth export.

Change 78889 on 2003/01/23 by llefebvr@llefebvr_r400_emu_montreal

Fixing underflow error in the interpolators.

Change 78854 on 2003/01/23 by llefebvr@llefebvr_r400_emu_montreal

Was overwriting the booleans at address 0 with the new ones. This is now working fine.

Change 78816 on 2003/01/23 by llefebvr@llefebvr_r400_emu_montreal

Forgot to add the CNDE,GT and DOT2ADD in the non-coissuable instructions list.

Change 78672 on 2003/01/22 by llefebvr@llefebvr_r400_emu_montreal

fixing the Frac problem with infinities.

Change 78584 on 2003/01/22 by llefebvr@llefebvr_r400_emu_montreal

I had a bug in the read register interface for RT fetch memory. It is now resolved.

Change 78324 on 2003/01/21 by llefebvr@llefebvr_r400_emu_montreal

fixing a bug with the _CONST opcodes and a memory export bug when exporting more than 1 pixel per block.

Change 77874 on 2003/01/20 by llefebvr@llefebvr_r400_emu_montreal

Fix for the scalar fract.

Change 77517 on 2003/01/17 by mmang@mmang_r400_win

Added new hardware accurate sqrt function code.

Change 77429 on 2003/01/17 by llefebvr@llefebvr_r400_emu_montreal

Fixing some FRAC precision issues along with MULADD problem with swizzle of SRCC.

Change 77415 on 2003/01/17 by lseiler@lseiler_r400_win_marlboro2

fixes for more warnings

Change 77293 on 2003/01/16 by lseiler@lseiler_r400_win_marlboro2

Changed XOR to != on lines 133, 513, and 515 to eliminate VCC warning message

Change 77210 on 2003/01/16 by llefebvr@llefebvr_r400_emu_montreal

Added a No_Serial bit to the alloc instruction. By seting this the SQ will not wait for texture data to have returned before allowing the alloc to go thru.

Change 76862 on 2003/01/15 by llefebvr@llefebvr_r400_emu_montreal

Tried to do the other opcodes as well where the denorm flushes where not done right. These include max,min,max4,trunc,frac and floor.

Change 76782 on 2003/01/15 by llefebvr@llefebvr_r400_emu_montreal

new muladd code and dot4 code. No functionnal changes cleanup only. Provided by Tom.

Change 75807 on 2003/01/10 by llefebvr@llefebvre_laptop_r400_emu

SX performance counters nomenclature change.
New dot product code (activated using RealDot Reg Key)

Change 75716 on 2003/01/10 by llefebvr@llefebvre_laptop_r400_emu

Fixing 2D shader bug where the sequencer was exectuting twice the last instruction.

Change 75569 on 2003/01/09 by llefebvr@llefebvre_laptop_r400_emu

Fixing exponent underflow in the mul part of the interpolators.

Change 75305 on 2003/01/08 by llefebvr@llefebvre_laptop_r400_emu

Now using the new mulAdd function.

Change 75032 on 2003/01/07 by llefebvr@llefebvre_laptop_r400_emu

fixing another problem with the cylindrical wrapping.

Change 74951 on 2003/01/07 by llefebvr@llefebvre_laptop_r400_emu

adding scalar _CONST opcodes to the emulator. Not tested.

Change 74888 on 2003/01/07 by llefebvr@llefebvre_laptop_r400_emu

Found a major bug in the SQ. This should fix the cyl wrapping problem. Needs to be verified with Jeff.

Change 74756 on 2003/01/06 by llefebvr@llefebvre_laptop_r400_emu

wasn't performing a denorm flush to zero on the input of the fract. Also doing it for the floor function as well now.

Change 74712 on 2003/01/06 by lseiler@lseiler_r400_win_marlboro2 eliminated VCC warning messages

Change 74397 on 2003/01/03 by llefebvr@llefebvr_r400_emu

Added Soft RBBM reset feature to the SQ.

Change 74359 on 2003/01/03 by llefebvr@llefebvr_r400_emu

Adding the additionnal restrictions on the MUL_PREV2 opcode.

Change 74297 on 2003/01/03 by sallen@sallen_r400_lin_marlboro

change all occurances of sbfloat<8,23,127> to proper mfloat<8,23,128>

Change 74163 on 2003/01/02 by llefebvr@llefebvr_r400_emu

I removed the input modifiers on previous scalar for the _PREV scalar opcodes per Tom's request.

Change 74144 on 2003/01/02 by llefebvr@llefebvr_r400_emu

There was a conflict between the counters of the SX and those of the SQ. I made the counters of the SX start at 0x30 to remove the conflict.

Change 73968 on 2002/12/31 by mmantor@FL_mmantorLT_r400_win

fixed for the vertex version of bad pipe to work. The sq changes corrected the steering of vertex data to the correct gpr's and the sx solved a hang when odd number of pipes were disabled and the vgt change corrected the pc deallocation generation circuit

Change 73536 on 2002/12/27 by llefebvr@llefebvre_laptop_r400_emu

New DOT code based on the newest muladd function.

Change 73502 on 2002/12/27 by llefebvr@llefebvre_laptop_r400_emu

Changing the XY import to the SP as per Tom Frinsinger's proposal.

Change 71733 on 2002/12/17 by llefebvr@llefebvre_laptop_r400_emu

New MulAdd has been added to the emulator. It is disabled by default. To enable please set NewMulAdd registry key or environement variable to 1.

Change 71681 on 2002/12/17 by llefebvr@llefebvre_laptop_r400_emu

Flushing denorms to 0 before the SET*,CDN* instructions before doing the test.

Change 71542 on 2002/12/16 by llefebvr@llefebvre_laptop_r400_emu

The SQ wasn't stuffing the vertices correctly in the GPRs when there where bad pipes.

Change 71161 on 2002/12/13 by llefebvr@llefebvre_laptop_r400_emu

Reverting the -ZERO change for the mul only. Now when doing a mul only +ZERO is added as source C.

Change 70456 on 2002/12/11 by llefebvr@llefebvre_laptop_r400_emu

Adding -0 for straight muls in HW accurate version instead of 0 for better precision.

Change 70403 on 2002/12/11 by llefebvr@llefebvre_laptop_r400_emu

fixing flushing to 0 the denorm BEFORE the NAN check...

Change 70159 on 2002/12/10 by llefebvr@llefebvre_laptop_r400_emu

Fixing overshifting in the interpolators adder.

Change 69667 on 2002/12/09 by llefebvr@llefebvre_laptop_r400_emu

fixing overshifting in the parameter_sub function of the SX.

Change 69637 on 2002/12/09 by llefebvr@llefebvre_laptop_r400_emu

Fixing underflow condition in the interpolators. Also flushing denorms for scalar table functions.

Change 69604 on 2002/12/09 by llefebvr@llefebvre_laptop_r400_emu

Backing out the change to the muladd that broke many regression tests.

Change 69176 on 2002/12/06 by llefebvr@llefebvre_laptop_r400_emu

adding debug info to the parameter subtract.

Change 69158 on 2002/12/06 by llefebvr@llefebvre_laptop_r400_emu

More HW precision fixes for the interpolators. Now matches perfectly HW for regular cases. Still a problem with very large numbers (but they are not used in the regression yet).

Change 68741 on 2002/12/05 by llefebvr@llefebvre_laptop_r400_emu

providing read path on the RBBM for the instruction store.

Change 68507 on 2002/12/04 by llefebvr@llefebvre_laptop_r400_emu

The emulator was off by 1 bit in the adder. Need to update golds... They should not missmatch by more than 1 LSB.

Change 68349 on 2002/12/04 by llefebvr@llefebvre_laptop_r400_emu

Error in the emulator implementation of TRUNC.

Change 68191 on 2002/12/03 by llefebvr@llefebvre_laptop_r400_emu

new implementation of the DOT product. Currently turned off by default. Need to set RealDOT to 1 in registry file to turn it on. Changed regular mulladd to reflect implementation changes as well.

Change 68039 on 2002/12/03 by llefebvr@llefebvr_r400_linux_marlboro

fixed a NaN propagation problem (the sign wasn't switched correctly)

Change 67086 on 2002/11/26 by llefebvr@llefebvr_r400_linux_marlboro

fixing typo

Change 67080 on 2002/11/26 by llefebvr@llefebvr_r400_linux_marlboro

fixing HW accurate boundary problems with SUB and range checking INF*0

Change 66324 on 2002/11/22 by llefebvr@llefebvre_laptop_r400_emu

Implemented the performance counters in the SQ and SX.

Change 65882 on 2002/11/21 by llefebvr@llefebvre_laptop_r400_emu

Fixing trunc opcode.

Change 65734 on 2002/11/20 by llefebvr@llefebvre_laptop_r400_emu

added performance counters in SQ.

Change 65556 on 2002/11/20 by llefebvr@llefebvre_laptop_r400_emu

Bug in the emulator dealing with large values being trunc or floored. This should fix a bunch of regression tests.

Change 65367 on 2002/11/19 by llefebvr@llefebvre_laptop_r400_emu

This wasn't hung. Just took a lot of time. I reduced the loop count to something more reasonable. Also changed the R400_NAN to FFC00000.

Change 65261 on 2002/11/19 by llefebvr@llefebvre_laptop_r400_emu

Fixing the FRAC opcode error with negative numbers.

Change 65199 on 2002/11/19 by llefebvr@llefebvre_laptop_r400_emu

There were two problems with this test:
1) The first pass pixel shader wasn't exporting anything thus the SX wasn't sending anything to the RBs hence the hang. You should never have a dummy pixel shader if there are pixel generated.
2) The time allowed to the test was too short. I incremented it to 10000 (was 1000).

I took the opportunity to do some code cleanup as well. Rerun other memory export tests to make sure I did not break anything.

Change 64957 on 2002/11/18 by llefebvr@llefebvre_laptop_r400_emu

Flipping face bit meaning in SQ (I had it backwards)

Change 64913 on 2002/11/18 by llefebvr@llefebvre_laptop_r400_emu

Fixing 2 HW accurate problems dealing with corner cases on RECIP and RECIPSQRT scalar opcodes.

Change 64795 on 2002/11/18 by mkelly@fl_mkelly_r400_win_laptop

* sq_block_model.cpp was not adding the buffer offset for centriods when centers and centriods are sent.
* fixed gold for r400sc_msaa_8_primtypes_01
* add r400sc_sp_sample_cntl_01 to SC regress_e to lock in this test and help minimize future debugging efforts.

Change 64467 on 2002/11/15 by llefebvr@llefebvre_laptop_r400_emu

Fixing cube bug in SP.

Change 64130 on 2002/11/14 by llefebvr@llefebvre_laptop_r400_emu

Fixing RT interpolation problem.

Change 63968 on 2002/11/14 by llefebvr@llefebvre_laptop_r400_emu

fixing FOG export bug in SX.

Change 63642 on 2002/11/13 by llefebvr@llefebvre_laptop_r400_emu

fixing full absolute constant problems in the emulator.

Change 63217 on 2002/11/11 by llefebvr@llefebvre_laptop_r400_emu

Fixing an error with loops in the SQ that caused it to loop indefinitely. Also adding some more predication tests.

Change 63153 on 2002/11/11 by llefebvr@llefebvre_laptop_r400_emu

fixing error that produces black output whenever using HW accurate mode.

Change 63113 on 2002/11/11 by llefebvr@llefebvre_laptop_r400_emu

Submiting an example of how to use the SQ generated counters.

Change 62908 on 2002/11/08 by llefebvr@llefebvre_laptop_r400_emu

removed the fix to float before the auto-counters. One must now subtract 2**23 before using them.

Change 62859 on 2002/11/08 by llefebvr@llefebvre_laptop_r400_emu

Fixed an exponent underflow problem in the HW accurate interpolators.

Change 62797 on 2002/11/08 by llefebvr@llefebvre_laptop_r400_emu

Added the write 0 fonctionnality to the PCs (bit 6 vector destination address).

Change 62709 on 2002/11/08 by llefebvr@llefebvre_laptop_r400_emu

Fixing the bug introduced by using the scalar address for the vector data in the GPRs.

Change 62558 on 2002/11/07 by askende@andi_crayola_emu

reversed the order in which the vector and scalar results are written in the GPRs.
Scalar first and Vector second.

Change 62485 on 2002/11/07 by llefebvr@llefebvre_laptop_r400_emu

Fixed a predication bug where if a pred set was done last it wouldn't count toward the optimisation.

Change 62057 on 2002/11/06 by llefebvr@llefebvre_laptop_r400_emu

Swapping the center/centroid sampling to reflect SC change.

Change 61970 on 2002/11/06 by llefebvr@llefebvre_laptop_r400_emu

Fixing the interpolators problem. This problem only shows up in tests that change the sampling modes (center/centroid).

Change 61027 on 2002/11/01 by llefebvr@llefebvre_laptop_r400_emu

Flipped GPR_MANAGEMENT fields to match CP.
Removed float to fix on the XY load.
Modified tests that used the feature accordingly.

Change 60854 on 2002/10/31 by llefebvr@llefebvre_laptop_r400_emu

Fixing the sending CENTERS only to the interpolators. Previously the emulator would only receive centers if centroids were also sent. Now you can send only the centers.

Change 60720 on 2002/10/31 by llefebvr@llefebvre_laptop_r400_emu

added 2 bits of precision to the muladd to match R300 vertex shader precision.

Change 60690 on 2002/10/31 by llefebvr@llefebvre_laptop_r400_emu

Problem with the predicate optimization resulting in this hang.

Change 60388 on 2002/10/30 by llefebvr@llefebvr_r400_linux_marlboro

fixing HW accurate bug on Linux.

Change 60279 on 2002/10/30 by llefebvr@llefebvre_laptop_r400_emu

Fixed the _prev opcodes. Now using the getSignbit instead of isNeg for NAN compatibility.

Change 60038 on 2002/10/29 by llefebvr@llefebvre_laptop_r400_emu

Renamed Real time parameter registers to be more explicit.
Implemented alpha to mask in the SX.
Implemented ALPHA_NO_BLEND in the SX.

Change 59911 on 2002/10/29 by mmantor@mmantor_r400_win

sq_block_model.cpp - made seperate line counters for loading of sq pixel vector data with a buffer count for the optimized quad transfers when quads from different rows can be sent togeather.
sx_block_mode.cpp/h seperated the quad fifo's so that even odd pairs are always read togeather to prevent reodering in the rb's
sc files added bad pipe to the packer and the optimized quad xfers

Change 59731 on 2002/10/28 by llefebvr@llefebvre_laptop_r400_emu

Memory overrun in the interpolators that was corrupting the state of the SQ. Also change the pred_clr opcode to write MAX_FLOAT instead of 5000 in the GPRs.

Change 59460 on 2002/10/25 by hartogs@fl_hartogs

Pipe Disable Change. This change creates a separate set of ROM_SP_disable bits for the vertex shader path and the pixel shader path.

Change 59417 on 2002/10/25 by llefebvr@llefebvre_laptop_r400_emu

Implemented the 4 IJ buffer performance optimization in the emulator.

Change 59135 on 2002/10/24 by llefebvr@llefebvre_laptop_r400_emu

Using getField instead of getReal for NAN detection in hope of fixing linux HW accurate bug.

Change 59077 on 2002/10/24 by llefebvr@llefebvre_laptop_r400_emu

Fixing a problem in the SQ RS management where the call return address wasn't save correctly in some cases. Fixes the advanced_test.

Change 58939 on 2002/10/23 by llefebvr@llefebvre_laptop_r400_emu

Underflow of the exponent fixed.

Change 58930 on 2002/10/23 by llefebvr@llefebvre_laptop_r400_emu

Fixes in HW accurate stuff.

Change 58812 on 2002/10/23 by llefebvr@llefebvre_laptop_r400_emu

Implemented Cube Opcode.
Added CNTX register for reading of the CST store using the RBBM bus for debug purposes.

Change 58676 on 2002/10/22 by llefebvr@llefebvre_laptop_r400_emu

Fixing a problem where the predicate bits were not set correctly wich caused problems with predicated texture fetches.

Change 58567 on 2002/10/22 by llefebvr@llefebvre_laptop_r400_emu

Pulsing the RB for CACHE flushes events in the VTX shader (and derivative flush events).

Change 58306 on 2002/10/21 by llefebvr@llefebvre_laptop_r400_emu

Fixed GradFills and Floor vector instruction.

Change 58287 on 2002/10/21 by llefebvr@llefebvre_laptop_r400_emu

Primlib was not passing NANDs correctly for ALU constants.
State changes are no more required for memory exports. I changed primlib to reflect this.

Change 58052 on 2002/10/18 by llefebvr@llefebvre_laptop_r400_emu

ABS modifier now applies to both constants.
Added apperture checks to prevent memory spills for memory exports.

Change 58005 on 2002/10/18 by llefebvr@llefebvre_laptop_r400_emu

Fixing r400vgt_vtx_export_very_very_simple_04 memory export test.

Change 58001 on 2002/10/18 by llefebvr@llefebvre_laptop_r400_emu

Various fixes to the assembler and the emulator to allow for debug exports.

Change 57713 on 2002/10/17 by llefebvr@llefebvre_laptop_r400_emu

added a debug address export destination

Change 57655 on 2002/10/17 by llefebvr@llefebvre_laptop_r400_emu

allowing PC allocs in the debugging mode.

Change 57534 on 2002/10/16 by llefebvr@llefebvre_laptop_r400_emu

allowing position allocs in debug mode.

Change 57526 on 2002/10/16 by llefebvr@llefebvre_laptop_r400_emu

fixing a debug export bug (wasn't letting position thru).

Change 57498 on 2002/10/16 by llefebvr@llefebvre_laptop_r400_emu

Added Predicate Exits support for loops in the emulator.

Change 57434 on 2002/10/16 by llefebvr@llefebvre_laptop_r400_emu

Not doing the range checking correctly for scalar ops. Was causing an error in 1 vgt test.

Change 57142 on 2002/10/15 by llefebvr@llefebvre_laptop_r400_emu

Added the _IEEE compliant opcodes to the scalar pipe. Updated Primlib in consequence.

Change 56880 on 2002/10/14 by llefebvr@llefebvre_laptop_r400_emu

Loops, jumps and calls are now using a 13 bit address which allows to jump and call and loop around any control flow addresses (does not requires to be even anymore).

Change 56697 on 2002/10/11 by llefebvr@llefebvre_laptop_r400_emu

Added a SQ_RB pulse interface for memory export synchronization.

Change 56607 on 2002/10/11 by llefebvr@llefebvre_laptop_r400_emu

Widened the event interfaces from 4 to 5 bits.

Change 56332 on 2002/10/10 by llefebvr@llefebvre_laptop_r400_emu

Missblending the fog in the color. Was shifting and shouldn't.

Change 56280 on 2002/10/10 by llefebvr@llefebvre_laptop_r400_emu

HW accuracy problem in interpolators fixed (HOS tests).
Abs is now executed before the negate.

Change 56093 on 2002/10/09 by llefebvr@llefebvre_laptop_r400_emu

Hw accurate bug fix when multiplying a number by 0 the fn was returning a very small but non-zero number.

Change 56036 on 2002/10/09 by llefebvr@llefebvre_laptop_r400_emu

Added the context ID to the SQ->TP interface. The name of the signal is SQ_TP_ctx_id. This is needed for multisampling resolves.

Change 56028 on 2002/10/09 by llefebvr@llefebvre_laptop_r400_emu

Fixed Nan*0 exeption case in the HW accurate mode of the muladd.

Change 55864 on 2002/10/08 by llefebvr@llefebvre_laptop_r400_emu

Fixed a hang in the SQ where if you didn't to a Tfect because of control flow the SQ would wiat forever after the TP. This enables the use of the new 2D pixel shader.

Change 55794 on 2002/10/08 by llefebvr@llefebvre_laptop_r400_emu

Added range check.
Also the test was clamping the values to infinities where clamped prior to the interface.

Change 55320 on 2002/10/04 by llefebvr@llefebvre_laptop_r400_emu

Updated the DOT2ADD opcode to meet new specification.
Checking in the test that reproduces the driver's setup for multiple textured triangles.

Change 55267 on 2002/10/04 by llefebvr@llefebvre_laptop_r400_emu

Fixed a bug introduced in vgt400_hos_PNT_01. The arbiter wasn't setting deallocation counts right.

Change 55126 on 2002/10/03 by llefebvr@llefebvre_laptop_r400_emu

multiple fixes for memory exports.

Change 54892 on 2002/10/02 by llefebvr@llefebvre_laptop_r400_emu

Fixed some memory export bugs.

Change 54819 on 2002/10/02 by llefebvr@llefebvre_laptop_r400_emu

fixing sub scalar.

Change 54748 on 2002/10/02 by llefebvr@llefebvre_laptop_r400_emu

Added a clear vertex ready counters in order to clean up the SQ counters on context changes.

Change 54680 on 2002/10/01 by llefebvr@llefebvre_laptop_r400_emu

Fixed problems with negative numbers in the muladd HW accurate routine.
Fixed HW accurate problem in the TP where it was not reading from the good vertex buffer index in HW accurate mode. This changes makes the whole mini-regression work in full HW accurate mode.

Change 54604 on 2002/10/01 by llefebvr@llefebvre_laptop_r400_emu

Fixing a HW accuracy bug in the interpolators. The mantissa of the A-B and A-C was subshifted by one.

Change 54515 on 2002/10/01 by llefebvr@llefebvre_laptop_r400_emu

Fixed HW accuracy problem in mull_add function where the wrong variable was used to shift the mantissa of the result of the mul prior to the add.

Change 54332 on 2002/09/30 by llefebvr@llefebvre_laptop_r400_emu

Fixed a bug the driver's compiler found where the SQ wasn't stopping on an EXEC_END in some cases.
Also fixed the arbiter's rules to make sure the new PC allocation scheme was respected.

Change 54149 on 2002/09/27 by llefebvr@llefebvre_laptop_r400_emu

Fixed the Sequencer to allow for early allocation of the PC. Allowing memory exports to occur at any point.

Change 54018 on 2002/09/27 by llefebvr@llefebvre_laptop_r400_emu

Fixed an error in the control flow machine where the last bit was set too early.

Change 53879 on 2002/09/26 by llefebvr@llefebvre_laptop_r400_emu

Fixed the SETs opcodes. Cleaned up the SX. Added an address buffer for memory exports.

Change 53821 on 2002/09/26 by llefebvr@llefebvre_laptop_r400_emu

Fixed the "last" bit warning that was not placed for the right condition.

Change 53670 on 2002/09/25 by llefebvr@llefebvre_laptop_r400_emu

The SQ was missreading the SQ_SAMPLING register wich was causing the interpolation of the data with the wrong IJs. Also fixed primlib to add the 1 parameter only SET and KILL scalar instructions.

Change 53592 on 2002/09/25 by llefebvr@llefebvre_laptop_r400_emu

corrected some minor errors in the vertex shader code for multiple state exports.

Change 50793 on 2002/09/11 by llefebvr@llefebvre_laptop_r400_emu

Fixing XY reads in the shader pipe.

Change 50772 on 2002/09/11 by llefebvr@llefebvre_laptop_r400_emu

Fixing the CP 2D test problem.

Change 50725 on 2002/09/11 by llefebvr@llefebvre_laptop_r400_emu

Fixed SQ dump file to output generated parameters as well.

Change 50697 on 2002/09/11 by llefebvr@llefebvre_laptop_r400_emu

Fixing the CP 2D bugs.

Change 50488 on 2002/09/10 by llefebvr@llefebvre_laptop_r400_emu

Added data dependant masks to the PCs.

Change 50464 on 2002/09/10 by llefebvr@llefebvre_laptop_r400_emu

Fixed HW accurate bug in the interpolators that caused inacuracies whenever the IJs where small.
Fixed the SP_SX interface to make it more HW accurate for the SX-RBRC testbench.
Made the SX more flexible for multiple pixel exports to the same location.

Change 50284 on 2002/09/09 by llefebvr@llefebvre_laptop_r400_emu

correction to the sub-normalized number interpretation of the IJs.

Change 50231 on 2002/09/09 by llefebvr@llefebvre_laptop_r400_emu

Corrected bad overflow handling in the HW accurate MULADD.

Change 50050 on 2002/09/06 by llefebvr@llefebvre_laptop_r400_emu

Fixed SWAP error that caused missmatches with gold images.
Added count reset capability in the SQ for multipass.
Added better comments for the PARAM_SHADE register field.
Fixed some HW accurate bugs.
Added new MOVA write back to GPRs feature.

Change 49961 on 2002/09/06 by ctaylor@fl_ctaylor_r400_dtwin_marlboro

Make MPASS_PIX_VEC_PER_PASS 20 bits since SQ auto-inc cannot be any bigger due to FltPt restrictions.
Add PA_SC_CNTL_STATUS.MPASS_OVERFLOW status flag.
Changed name and added new event for controlling MPASS pixel shaders
and SQ vertex and pixel counters.
Added SC->CP VizQuery and MP PixShader dumps.
Made window_offset register fields 15 bits instead of 16.
Added SC MP PixShader logic.
Added new signal to SC->SQ interface to prevent incrementation of pixel count for "discarded" MP Pix Shader Pixel Vectors.
Fixed MSAA bug where samples 4-7 were not being set to 0 when MSAA was disabled. Fixed both EMU and RTL.

Change 49761 on 2002/09/05 by sallen@sallen_r400_lin_marlboro

ferret: add some interfaces for gc comparitors

Change 49690 on 2002/09/05 by llefebvr@llefebvre_laptop_r400_emu

Added HW accurate scalar operations but for the SQRT. Tested Log and Exp to some extent. Using Khan 6X Scalar implementation.

Change 49621 on 2002/09/05 by llefebvr@llefebvre_laptop_r400_emu

Added GEN_INDEX_VTX field to the SQ_PROGRAM_CNTL register.
Changed name of GEN_INDEX field to GEN_INDEX_PIX.

Change 49556 on 2002/09/04 by llefebvr@llefebvre_laptop_r400_emu

Fixed another Linux compilation error

Change 49549 on 2002/09/04 by llefebvr@llefebvre_laptop_r400_emu

removed some warnings and a Linux error.

Change 49540 on 2002/09/04 by llefebvr@llefebvre_laptop_r400_emu

Updated the SX->SQ position ready interface to add one more bit.
Added code for the HW accurate scalar pipe (not used yet).

Change 49374 on 2002/09/03 by mmantor@mmantor_r400_win

added sc output of centers and xy data. Also passed the xy data on the ij bus between the SC and SQ

Change 49353 on 2002/09/03 by llefebvr@llefebvre_laptop_r400_emu

Added Idle0 and Idle1_7 functions for SQ idle status report.

Change 49276 on 2002/09/03 by llefebvr@llefebvre_laptop_r400_emu

Added the RECIP_FF and RECIPSQ_FF scalar opcodes.
Reorganized the opcode enums to match most recent HW.
Added a more detailed description of the EO_RT control register.
Updated primlib to reflect the opcode changes.

Change 49038 on 2002/08/30 by llefebvr@llefebvre_laptop_r400_emu

Added debug dumps for interpolator inputs and vertex shader exports to parameter caches.

Change 48886 on 2002/08/29 by llefebvr@llefebvre_laptop_r400_emu

Implemented the -MAX_FLOAT changes for the MUL_PREV2, LOG and EXP opcodes.
Implemented the HW accurate version of DOT3,DOT4 and DOT2ADD. All vector opcodes are now HW accurate.

Change 48796 on 2002/08/29 by llefebvr@llefebvre_laptop_r400_emu

Changed to MUL_PREV2 instruction to output -infinity also if SrcC is < 0 (per Andy Gruber's request).

Change 48770 on 2002/08/29 by llefebvr@llefebvre_laptop_r400_emu

Implemented the new "9 bits constant" mode. If set we use the constant address as a real absolute address in the constant store (we do not add the VS_CONST_BASE or PS_CONST_BASE to the address specified in the instruction). Only available for constant A.

Change 48237 on 2002/08/27 by llefebvr@llefebvre_laptop_r400_emu

Added Event filtering in the SQ.

Added better Idle Status reporting in the SQ that now takes the VGT interface into account (previous version was not taking the staging registers into account just the interface).

Change 48140 on 2002/08/26 by llefebvr@llefebvre_laptop_r400_emu

Fixed 3 bugs in the HW accurate interpolators:
1) Detection of I and J == 0 was wrong
2) Detection of A == 0 was wrong
3) Denormalization of second paramter was done in the inverse direction of what it should have been.
This fixes both primlib_template_simple_triangle.cpp and milestone_tri.cpp tests

Change 48080 on 2002/08/26 by llefebvr@llefebvre_laptop_r400_emu

There was an indexing problem with the writting of the STs in the GPRs.

Change 47880 on 2002/08/23 by llefebvr@llefebvre_laptop_r400_emu

This is the real fix for bug 299. The other one created nasty side effects in the VGT.

Change 47751 on 2002/08/23 by llefebvr@llefebvre_laptop_r400_emu

There was an error in the SQ that made it wrap around when the SQ_PS/VS_CONST.BASE value was too high. This is fixed.

Change 47592 on 2002/08/22 by llefebvr@llefebvre_laptop_r400_emu

NewVector was not handled correclty when not associated with a pixel vector.

Change 46827 on 2002/08/19 by llefebvr@llefebvre_laptop_r400_emu

For SX quad Fifo reasons we are now forcing memory exports to occur in order.

Change 46728 on 2002/08/19 by llefebvr@llefebvre_laptop_r400_emu

Fixed another MAX GPR bug in the Vertex shader.

Change 46550 on 2002/08/16 by llefebvr@llefebvre_laptop_r400_emu

Added more checks for the Idle SQ function.

Change 46516 on 2002/08/16 by llefebvr@llefebvre_laptop_r400_emu

Fixed the maximum GPR (64) allocation problem. One can now run shaders that use 64 GPRs. The meaning of the VS/PS_NUM_REG has changed it now represents the INDEX OF THE MAXIMUM GPR NUMBER. This change will break the HW.

Change 46315 on 2002/08/15 by llefebvr@llefebvre_laptop_r400_emu

The SQ->SC dump file was not showing all transactions that occured. This is now corrected. Also changed the default 2D shader directory (if your $BRANCH and $ROOT variables aren't set).

Change 46246 on 2002/08/15 by llefebvr@llefebvre_laptop_r400_emu

Fixed an instruction wrapping problem in the SQ

Change 46140 on 2002/08/14 by llefebvr@llefebvre_laptop_r400_emu

Changed the MASK mnemonics to KILL.
Added DST opcode.
Added MUL_PREV2 opcode.
Reordered the opcodes in primlib and SP.
Implemented the new KILL and SET SCALAR opcodes, they are now all comparing the ALPHA channel to 0.0f (instead of comapring against the RED channel).

Change 46049 on 2002/08/14 by sallen@sallen_r400_lin_marlboro

ferret: change bool/boolean to boolnumber & test
change pix_mask to TP_SP_pix_mask, etc
more sqspsx tweaks

Change 46036 on 2002/08/14 by llefebvr@llefebvre_laptop_r400_emu

Fixed the clamping problem. Note however that now there is no triangle drawn in the test because the clamping causes the triangle to be degenerate.

Change 45961 on 2002/08/14 by llefebvr@llefebvre_laptop_r400_emu

Now allowing one position vector to be exported in memory export mode to "ping" the PA that everything is done.

Change 45752 on 2002/08/13 by llefebvr@llefebvre_laptop_r400_emu

Signal name change on the TP_SQ interface and backup of the not yet working fully memory export in the SX.

Change 45388 on 2002/08/12 by llefebvr@llefebvre_laptop_r400_emu

Changed the SX allocation strategy to better match the HW in MRT cases. The new allocation scheme puts all MRTs in a single block instead of interleaving the MRTs together like the previous implemetation.

Change 45216 on 2002/08/09 by sallen@sallen_r400_lin_marlboro

tpc_t4 changes for TP_SQ interface

Change 45200 on 2002/08/09 by llefebvr@llefebvre_laptop_r400_emu

Corrected a bug in MUL_PREV,ADD_PREV and SUB_PREV wich where using incorect source operands.

Change 45127 on 2002/08/09 by llefebvr@llefebvre_laptop_r400_emu

The SQ now uses the information in state registers to allocate regular PS/VS shaders. The size field is ONLY USED for memory writes.

Change 45015 on 2002/08/08 by llefebvr@llefebvre_laptop_r400_emu

Fixed the fog blend logic in the SP and added SETNEs missing opcode.

Change 44461 on 2002/08/06 by llefebvr@llefebvre_laptop_r400_emu

Fixed a problem in the SQ related to the 2D shader. Now the SQ runs even when the END flag is not associated with any instruction (it adds a NOP to EXEC_ENDs with a count of 0).

Change 44350 on 2002/08/06 by llefebvr@llefebvre_laptop_r400_emu

There was a bug in the SQ that made the last iteration of a loop read invalid loop indexes. This is now fixed.

Change 44321 on 2002/08/05 by askende@askende_r400_linux_emu

added hardware bit accurate implementation for some of the vector instructions

Change 44252 on 2002/08/05 by llefebvr@llefebvre_laptop_r400_emu

New counter based predication scheme.

Change 44232 on 2002/08/05 by llefebvr@llefebvre_laptop_r400_emu

Fixed a parameter generation bug in SQ wich was causing the emulator to crash for test r400su_point_sprite_01.

Change 44054 on 2002/08/02 by llefebvr@llefebvre_laptop_r400_emu

Fixed the emulator to add automatically the generated parameters to the number of interpolated parameters in order not to screw up the parameter cache pointer computation in the VGT when generating parameters.

Change 44026 on 2002/08/02 by llefebvr@llefebvre_laptop_r400_emu

Implemented the new parameter generation scheme. See SQ spec for details.

Change 43826 on 2002/08/01 by askende@askende_r400_linux_emu mods related to matching the hardware accuracy

Change 43756 on 2002/08/01 by askende@askende_r400_linux_emu

new mod on the interpolators to get it to match hardware

Change 43742 on 2002/08/01 by sallen@sallen_r400_lin_marlboro

ferret: move HW related files from test_lib/tools/ferret to parts_lib/src/test/ferretutils
make associated changes
add interface changes for tp4_tc testbench
make some tweaks for gc testbench

Change 43661 on 2002/08/01 by llefebvr@llefebvre_laptop_r400_emu

dumping in Hex format

Change 43394 on 2002/07/31 by llefebvr@llefebvre_laptop_r400_emu

HW accurate interpolators in the SQ. To turn on, set HardwareAccurate environment variable to 1 (or HKEY_LOCAL_MACHINE\\SOFTWARE\\ATI Technologies\\Emulator\\Debug\\HardwareAccurate to 1 if in windows). Off by default.

Change 43234 on 2002/07/30 by llefebvr@llefebvre_laptop_r400_emu

The LOD correction bits are now correctly propagated to the TP.

Change 43129 on 2002/07/30 by llefebvr@llefebvre_laptop_r400_emu

Initialized the staging registers to prevent the NAN error message in the interpolators to be issued without a valid reason.
Corrected a typo wich prevented the control flow booleans to be written correctly.

Change 42994 on 2002/07/29 by llefebvr@llefebvre_laptop_r400_emu

Added the register to disable HW detection of PV/PS.
Fixed infinite looping problem when ending a shader on a NOP.
Fixed the BW jump problem.

Change 41911 on 2002/07/23 by hwise@fl_hwise_r400_win

Fixed type casting bug when getting eventId from vgt_sq_verts_data.VGT_SQ_vsisr_data[0]

Change 41828 on 2002/07/22 by mmantor@mmantor_r400_win

connected event and event_id into the pa_sc interface
fixed misc dump files for test bench working
added lod_correct values to sc_sq interface

Change 41596 on 2002/07/19 by llefebvr@llefebvre_laptop_r400_emu

Corrected the VGT->SQ event interface. Corrected the GFX_COPY_STATE problem.

Change 41029 on 2002/07/17 by llefebvr@llefebvre_laptop_r400_emu

Placed the hooks for HW accurate interpolation (not used yet)...

Change 40887 on 2002/07/16 by llefebvr@llefebvre_laptop_r400_emu

Implemented the new bad pipe interface between the VGT and the SQ. The VGT doesn't pad anymore and so the SQ is responsible to "jump" over a bad pipe when filling the reservation stations.

Change 40867 on 2002/07/16 by llefebvr@llefebvre_laptop_r400_emu

Minor changes to the addressing routines of the SQ in preparation for multipass implemetation.

Change 40695 on 2002/07/15 by llefebvr@llefebvre_laptop_r400_emu

Fixed the event interface in the SQ where it was using a NULL pointer as a valid mask bit field wich was then crashing the numerical library. This changes should fix both the vertex and the pixel event interface. The primlib_vgt_event_initiator.cpp now works correctly.

Change 39443 on 2002/07/10 by hwise@fl_hwise_r400_win

CP, SQ, and Primlib Updates for PM4 packet format changes

Packet Changes
1) SET_CONSTANT
a) Updated CONST_ID field to include "booleans" and "loop"
b) Removed ALU and Texture constant grouping and now use
DWORD offset in packet
c) Removed CONST_WRITE_ENABLE field from ordinal 1 (the ALU and
TEX write enables are now set with the
LOAD_CONSTANT_CONTEXT
packet)
d) Replaced CONST_INDEX with CONST_OFFSET because meaning changed
(offset refers to dword offset and index used to refer to
constant group. Also real-time vs. non-real-time is implied
by the queue from which the packet is read rather than being
encoded in the index)
2) LOAD_CONSTANT_CONTEXT
a) Updated CONST_ID field to include "booleans" and "loop"
b) Removed ALU and Texture constant grouping and now use
DWORD offset in packet
c) Replaced CONST_INDEX with CONST_OFFSET because meaning changed

(offset refers to dword offset and index used to refer to
constant group. Also real-time vs. non-real-time is implied
by the queue from which the packet is read rather than being
encoded in the index)
d) Replaced NUM_CONSTANTS ordinal with NUM_DWORDS where number
of constants had to be converted into dwords via microcode
e) Send CONST_PREFETCH packet to the Microengine rather than
the LOAD_CONSTANT_CONTEXT
3) CONST_PREFETCH
a) This is a new packet
4) DRAW_INDX
a) Removed INDEX offset ordinal from packet (this is now state
that must be updated prior to the draw packet)

CP Pre-Fetch Parser
1) Added state to hold ALU and Texture write enables used when
parsing SET_CONSTANT and LOAD_CONSTANT_CONTEXT packets
2) When parsing LOAD_CONSTANT_CONTEXT packet, send CONST_PREFETCH
packet to the Microengine rather than the LOAD_CONSTANT_CONTEXT

CP Microengine
1) Updated logic for reporting Microengine IDLE state
2) Added a lot of support logic for multi-context
3) Fixed IP stack popping bug when returning from subroutine

SQ Register Write Decode Logic
1) SQ no longer pads 2 dwords between groups of 6 dwords when
decoding register addresses

Primlib updates for new packet formats
1) Added BOOLEAN_CONSTANT, and LOOP_CONSTANT to enum
CONSTANT_TYPE
in render_engine.h
2) Updated RENDER_ENGINE member functions to support new PM4
SET_CONSTANT packet format
a) RENDER_ENGINE::Load_Alu_Constants()
b) RENDER_ENGINE::Load_Texture_Constants()
c) RENDER_ENGINE::Open_Set_Constant_Packet()
3) Updated member function RENDER_ENGINE::Load_Draw_Command() to
support new 3D_DRAW_INDX_2 and DRAW_INDX PM4 packet formats

CP full-chip tests
1) Updated packet creation within tests to match the PM4 spec
changes

Change 39130 on 2002/07/09 by llefebvr@llefebvre_laptop_r400_emu

Fixed scalar trunc, floor and frac opcodes.

Change 38783 on 2002/07/08 by llefebvr@llefebvre_laptop_r400_emu

Changed the TP_SQ/SQ_TP interface rs_line name to thread_id to match the HW. Made the corresponding ferret changes.

Change 38277 on 2002/07/05 by llefebvr@llefebvre_laptop_r400_emu

Implemented correctly lines and points in the SQ.

Change 37942 on 2002/07/03 by llefebvr@llefebvre_laptop_r400_emu

New interpolation scheme for ST generation for points and lines. Also made the correct changes for flat shading of points and lines.

Change 37862 on 2002/07/03 by llefebvr@llefebvre_laptop_r400_emu

Changed the interpolation equation from C+I*(B-C)+J*(A-C) to A+I*(B-A)+J*(C-A) to match with the changes that occured in the SU to do lines and points cleanly.

Change 37822 on 2002/07/03 by llefebvr@llefebvre_laptop_r400_emu

Added Call stack size checks in the SQ.

Change 35516 on 2002/06/21 by llefebvr@llefebvre_laptop_r400_emu

Solidified the interpolators in order to make the HW accurate transition easier.

Change 35484 on 2002/06/21 by llefebvr@llefebvre_laptop_r400_emu

Predication optimization problem fixed in Sq.

Change 35105 on 2002/06/20 by llefebvr@llefebvre_laptop_r400_emu

corrected the sq tests to use the new primlib include.

Change 34940 on 2002/06/19 by kmahler@kmahler_r400_win_devel_views

Shader assembler and SQ block changes to support new ALU packing including new absolute modifier in source operand.

This also includes syntax checking for new restrictions on export register usage. That is, the destination operands of both the vector and scalar operations must be an export if one is an export. The building of the implicit NOP had to be changed to support this restriction. Syntax checking was added to ensure that only one constant register uses an absolute modifier. The co-issue delimitor must now be "|" for shader versions not equal to "1.0". Version "1.0" may use either the new delimitor, "|", or the old one, ":".

Change 34483 on 2002/06/17 by llefebvr@llefebvre_laptop_r400_emu

Implemented scalar exports

Change 34451 on 2002/06/17 by llefebvr@llefebvre_laptop_r400_emu

Changed the encoding of the SQ_PROGRAM_CNTL.VS_EXPORT_CNT register. Now a value of 0 means 1 parameter and so forth. Everyone reading the register must now add 1 to have the correct number of parameters. This allows to cover the range 1->16 with only 4 bits.

Change 34419 on 2002/06/17 by llefebvr@llefebvre_laptop_r400_emu

Added MOVA opcode to the vector path and MOVA_TRUNC opcode to the scalar path...

Change 34233 on 2002/06/14 by llefebvr@llefebvre_laptop_r400_emu

Implemented all the new predicate set instructions (vector and scalar).

Change 34172 on 2002/06/14 by llefebvr@llefebvre_laptop_r400_emu

Added/Removed vector and scalar opcodes to comply with the new ALU format as specified by Andy Gruber.

Change 33840 on 2002/06/13 by llefebvr@llefebvre_laptop_r400_emu

implemented more scalar opcodes.

Change 33485 on 2002/06/12 by llefebvr@llefebvre_laptop_r400_emu

Implemented most of the remaining vector opcodes. Including pixel kills. The pred_set are still not implemented since they are still changing.

Change 33165 on 2002/06/11 by llefebvr@llefebvre_laptop_r400_emu

Fixed parameter cache addressing error when exporting more than one color in the VS.

Change 32964 on 2002/06/10 by llefebvr@llefebvre_laptop_r400_emu

Fix for r400vgt_hos_cubic_pos_pnt_discrete_01 crashing when vertexTail == 0.

Change 32633 on 2002/06/07 by llefebvr@llefebvre_laptop_r400_emu

added an assert for NANs in the interpolators.

Change 31997 on 2002/06/05 by llefebvr@llefebvre_laptop_r400_emu

Added SQ_ prefix to allocation_type enum because of a name clash in windows.h

Change 31843 on 2002/06/04 by llefebvr@llefebvre_laptop_r400_emu

Added CONDITIONAL_EXEC_PRED_NO_STALL and CONDITIONAL_EXEC_PRED_NO_STALL_END control flow opcodes to the SQ per Andy Gruber's request.

Change 31717 on 2002/06/04 by llefebvr@llefebvre_laptop_r400_emu

Fixed the ABSOLUTE/RELATIVE name clash in the SQ enum by changing the names to ABSOLUTE_ADDR/RELATIVE_ADDR. Fixed some warnings and fixed the SQ_SC_dec_cnt problem where the SQ should have pulsed twice the SC on a new vector but it was only pulsing once.

Change 31527 on 2002/06/03 by llefebvr@llefebvre_laptop_r400_emu

Updated the objParser. Added a bypass of the interpolators if all parameters of the primitive are the same.

Change 31254 on 2002/05/31 by askende@andi_crayola_emu_w

fixed a clamping related bug. Numbers greater than 1.0 weren't being clamped when clamping is on.

Change 31115 on 2002/05/31 by llefebvr@llefebvre_laptop_r400_emu

Added safety checks in the sq vertex processing

Change 30964 on 2002/05/30 by llefebvr@llefebvre_laptop_r400_emu

Implemented the Event interface in the SQ (for vertices).

Change 30942 on 2002/05/30 by llefebvr@llefebvre_laptop_r400_emu

updated the VGT->SQ interface (and corresponding blocks) to match HW in name and size and add the Event field.

Change 30570 on 2002/05/29 by llefebvr@llefebvre_laptop_r400_emu

fixed a parameter cache deallocation synchronization problem

Change 30311 on 2002/05/28 by llefebvr@llefebvre_laptop_r400_emu

removed useless code...

Change 30310 on 2002/05/28 by llefebvr@llefebvre_laptop_r400_emu

The CoissuedInstruction flag was not reset properly in sq_alu.cpp. If one mulAdd was done in the shader the scalar path was turned off for the remainder of the shader program. This is now corrected.

Change 30106 on 2002/05/26 by ctaylor@fl_ctaylor_r400_dtwin_marlboro

Remove FloatParts usage. concerned about speed/memory rqmts.

Change 30068 on 2002/05/24 by ctaylor@fl_ctaylor_r400_dtwin_marlboro

Update SC and SP to use denormalized SE4M20 (exp bias of 6) for IJ data.

Change 29937 on 2002/05/24 by llefebvr@llefebvre_laptop_r400_emu

Fixed a event related problem that would have hanged the sequencer in the future (when events are turned on).

Change 29850 on 2002/05/24 by llefebvr@llefebvre_laptop_r400_emu

Changed the SQ_PROGRAM_CNTL.VS_EXPORT_MODE register field to use this enumeration instead:

0: Position (1 vector).
1: Position (2 vectors) Do not use second position export.
2: Position (2 vectors) Use point sprite size.
3: Position (2 vectors) Use edge flags.
4: Position (2 vectors) Use kill flags.
5: Position (2 vectors) Use point sprite size and kill flags.
6: Position (2 vectors) Use edge flags and kill flags.
7: Multipass.

Changed all emulator files (and primlib) to match with the new setting. No impact on the RTL (not using the feature yet).

Change 29678 on 2002/05/23 by llefebvr@llefebvre_laptop_r400_emu

Implemented the debug features of the SQ (untested and unused at the moment).

Change 29461 on 2002/05/22 by llefebvr@llefebvre_laptop_r400_emu

Corrected the "opcode 15 not supported" error from the primlib_tex test.

Change 29385 on 2002/05/21 by llefebvr@llefebvre_laptop_r400_emu

Fixed an arbitration problem in arbiter where two exporting vectors could have the same thread id (causing a hang in the SX).

Change 29075 on 2002/05/20 by llefebvr@llefebvre_laptop_r400_emu

Fixed a problem in the vertex input engine of the SQ where the SQ was invalidating the last vertex sent if the continued signal was asserted.

Change 29039 on 2002/05/20 by llefebvr@llefebvre_laptop_r400_emu

Added support for RECIPSQ, LOGs, EXPs. Software implementation only using the C standard funtions.

Change 28843 on 2002/05/17 by llefebvr@llefebvre_laptop_r400_emu

Documented all control flow instruction types in sq.blk (and sq.desc).

Change 28778 on 2002/05/17 by askende@andi_crayola_emu_w

fixed the swizzle logic

Change 28375 on 2002/05/16 by llefebvr@llefebvre_laptop_r400_emu

Changed the Allocate_size field of the Alloc instruction to match the encoding of the HW.
The encoding is now:
00 -> 1 buffer
01 -> 2 buffers
...
15 -> 16 buffers

Change 28208 on 2002/05/15 by llefebvr@llefebvre_laptop_r400_emu

Added the enumeration fields in autoreg for both ALU and Control Flow instructions. Not a functionnal change.

Change 28110 on 2002/05/15 by llefebvr@llefebvre_laptop_r400_emu

Replaced the reduce precision delat interpolation scheme with full precision interpolation.

Change 27976 on 2002/05/14 by llefebvr@llefebvre_laptop_r400_emu

New SQ register map.
Added EXEC_END,CEXEC_END, CPEXEC_END. Removed the END instruction.

Change 27875 on 2002/05/14 by jhoule@jhoule_r400_win_marlboro

Sequencer was calling GetNewTP_SQ_pix_mask instead of GetTP_SQ_pix_mask.
TP returns correct masks now (was previously all 1s to be conservative).

Change 27413 on 2002/05/10 by llefebvr@llefebvre_laptop_r400_emu

Added texture pipe predication support.

Change 27370 on 2002/05/10 by jhoule@jhoule_r400_win_marlboro

Added pix_mask support (always returns 1111 from TP).

Added srcSwizzle (TInstrPacked) called from the SQ (arbiter) in order to keep 3 channels SP->TP path.
Adapted tp_dumps to represent all of those.

Change 27266 on 2002/05/09 by vliu@vliu_r400_cnvliu100_win_cvd

Fixed the compile error after remove of the mc block
  - everyone, please do a clobber under emu_lib and rebuild
    since the ar_*.* files under ar_code will still be there if not removed

Change 27260 on 2002/05/09 by llefebvr@llefebvre_laptop_r400_emu

Added the pixel masks to the TP->SQ interface. Implemented the JUMP opcode.

Change 27218 on 2002/05/09 by llefebvr@llefebvre_laptop_r400_emu

Added the W field to the SP->TP interface.
Implemented CALL, LOOP and COND opcodes in the control flow machine.

Change 27011 on 2002/05/08 by llefebvr@llefebvre_laptop_r400_emu

Changed the number of bits of the event_id field from 2 to 4 in the sequencer and the corresponding interface per Mike Mantor's request.

Change 26974 on 2002/05/08 by llefebvr@llefebvre_laptop_r400_emu

Added constant relative indexing via address register support and predication support (vector 0 only) in the SP. The pred_set and mova instructions are still not implemented so this is of little use right now.

Change 26955 on 2002/05/08 by llefebvr@llefebvre_laptop_r400_emu

Added relative constant and GPR addressing capabilities to the SQ.

Change 26797 on 2002/05/07 by llefebvr@llefebvre_laptop_r400_emu

Changed the mbfloats in the interfaces for mfloats in order to use IEEE floats instead of doubles in the SP, SX, and RBs. Added the control_flow_store to the SQ.

Change 26750 on 2002/05/07 by llefebvr@llefebvre_laptop_r400_emu

Corrected a problem in the SQ when loading multiple stages in the vertex staging registers.

Change 26569 on 2002/05/06 by llefebvr@llefebvre_laptop_r400_emu

Added readState and WriteState utility functions to the control flow machine in the SQ.

Change 26522 on 2002/05/06 by llefebvr@llefebvre_laptop_r400_emu

There was a problem in the SQ whenever there where two EXEC instructions in a row. This is now fixed. It fixed the regression and hopefully will fix all other tests using the same template.

Change 26466 on 2002/05/06 by llefebvr@llefebvr_r400_linux_marlboro

fixed linux compilation errors

Change 26365 on 2002/05/03 by llefebvr@llefebvre_laptop_r400_emu

This is the new control flow sequencer. Expect things to be a bit unstable while this major change settles in. I know I broke 1 regression test (r400vgt_index_size_01) but the integration took so long that I decided to check the change in anyways and fix the problem from the TOTT. Sorry for the inconvenience.

Change 26344 on 2002/05/03 by hartogs@fl_hartogs

Added a count the VGT_SQ interface to verify the proper operation
of the interface. The added code is #ifdef'ed out and is not
a functional change.

Change 24998 on 2002/04/25 by llefebvr@llefebvre_laptop_r400_emu

Small correction in the SQ again because of a bug in the flat shading control code.

Change 24929 on 2002/04/25 by llefebvr@llefebvre_laptop_r400_emu

Fix for the flat shading (there was a shifting problem in the SQ)

Change 23847 on 2002/04/18 by llefebvr@llefebvre_laptop_r400_emu

Fixed a small bug in the SC that made the SQ stall whenever the IJ buffers where filled up.

Change 23698 on 2002/04/17 by llefebvr@llefebvre_laptop_r400_emu

Updated the SQ->TP and TP->SQ interfaces to match the HW in names.

Change 23236 on 2002/04/15 by llefebvr@llefebvre_laptop_r400_emu

Added an error check in the SQ to report an error when trying to dealocate an empty parameter cache.

Change 23080 on 2002/04/12 by rramsey@RRAMSEY_P4_r400_win

Fix bug in interp bb logic
    Add commentline dump to sc dump class
    Add state dump to sc

Change SQ input to allow new_vector and pc_dealloc to happen
  on the same quad from the SC. Laurent L. needs to verify that
  this change is OK, but it fixes the scissor_rect_04 hang and
  passes chip regression.

Change 23010 on 2002/04/12 by llefebvr@llefebvre_laptop_r400_emu

Added the #define in vgtout that makes the deallocation of the PC occur on the last primitive instead of having to add another null prim.

Change 22994 on 2002/04/12 by llefebvr@llefebvre_laptop_r400_emu

Added the SQ->SC interface dump and corrected a small bug in the other interface dumps of the SQ,SX that made the dumps slightly out of sync with what was actually sent over the interfaces.

Change 22976 on 2002/04/11 by hartogs@fl_hartogs

* sq_block_model.cpp --> Flipped #if switch to new dealloc mode. Fixed variable name typo.
* interconnect.cpp, interconnect.h --> Added DumpComment() routine to the Interconnect
    class to allow comments in the dump file. I used this to print packet boundary
    comments into the VgtPaClipP dump for easier cross-checking with the other dumps.
* vgt_pa_if.h --> Changed the "dealloc_slot" field from a 'Bool' to an unsigned long.
* vgtout.h, vgtout.cpp --> Coded new dealloc mode.
* vgt_block_model.cpp --> Changed deallocate_slot from bool to ulong.
* vgt_pa_clip_prim.h --> Changed deallocate_slot from bool to ulong.
* For this check-in, the dealloc value can be greater than one, however, the dealloc
    does NOT occur on the prim using the verts.

Change 22823 on 2002/04/11 by llefebvr@llefebvre_laptop_r400_emu

Added the SC_SQ_provok_vtx to the SC->SQ interface (Randy you must set the correct value if you want flat shading to work).

Added the following interface dumps to the SX:
SX->RB
SX->RB_quads

Added the two ring wrapping modes for the SQ instruction store.

Change 22547 on 2002/04/09 by llefebvr@llefebvre_laptop_r400_emu

Added the capability to dump interfaces to a file to the SQ and SX blocks.

The SQ can now dump the following interfaces:
SQ->SX

SP->SX
SQ->TP

The SX can now dump the following interface:
SX->PA

Change 22459 on 2002/04/09 by llefebvr@llefebvre_laptop_r400_emu

Added the auto-count to the vertex shaders. The auto-generated counter is loaded in R2.x if SQ_IMPORTS_EXPORTS.GEN_INDEX is set.

Change 22449 on 2002/04/09 by llefebvr@llefebvre_laptop_r400_emu

Added the new PC dealloc scheme (the old one is still there and is the one active for now).

Change 22354 on 2002/04/08 by llefebvr@llefebvre_laptop_r400_emu

Added the cylindrical wrapping logic in the interpolators.

Change 22138 on 2002/04/05 by llefebvr@llefebvre_laptop_r400_emu

Modified the SQ_IMPORTS_EXPROTS.VS_EXPORT_MODE register to reflect the proposed change of the PA review.

Change 22048 on 2002/04/05 by llefebvr@llefebvre_laptop_r400_emu

Added the SQ_CP event interface.
Added support for centers/centroid sampling.
Put provok vertex to 2 if set to last in state (TEMP/ CLAY will confirm).

Change 21910 on 2002/04/04 by llefebvr@llefebvre_laptop_r400_emu

Implemented:
1) Real time interpolation.
2) Auto counters for pixel vectors.
3) Flat Shading.
4) Faceness buffers
5) XY buffers
6) Sprite texture coordinate generation

Change 21673 on 2002/04/03 by llefebvr@llefebvre_laptop_r400_emu

Implemented Real time and multi-state management in the SQ (not tested)

Change 21630 on 2002/04/03 by llefebvr@llefebvre_laptop_r400_emu

Added the setContextNumber() functions wherever it was necessary in both the SQ and the SX blocks.

Change 21539 on 2002/04/03 by llefebvr@llefebvre_laptop_r400_emu

Added the event pipelining from the CP (interface back to the CP not done yet).

Change 21480 on 2002/04/02 by rramsey@RRAMSEY_P4_r400_win

Remove OLD_SC stuff from PA and make sure all the sc interface includes are in the new sc directory and being called from there

Change 21275 on 2002/04/01 by llefebvr@llefebvre_laptop_r400_emu

Now using the get functions on all context register reads from the SX and the SQ.

Change 20682 on 2002/03/27 by llefebvr@llefebvre_laptop_r400_emu

New interfaces between SC,SQ,SP and SX are now implemented and functionnal.

Change 20483 on 2002/03/26 by jhoule@jhoule_r400_win_marlboro

Implemented new Const/Instr classes.
Integrated with Primlib so that it passes regression.
TFetches don't seem to pass, but VFetches are OK.

Numerous changes:
- Uses const & and plain & for read/write in SQ classes
- Uses accessor methods for set() and get()
- Created new Instr/Const classes, with inheritance (no more unions)
- Created *Packed classes, which are used in the SQ

Change 20198 on 2002/03/22 by llefebvr@llefebvre_laptop_r400_emu

Added masking support for the Parameter caches. Added support to write the color buffers out of order to the Export buffer.

Change 19923 on 2002/03/21 by llefebvr@llefebvre_laptop_r400_emu

Added pixel kill functionnality in the SX. Fixed the SP->SX interface names to match those of the HW.

Change 19654 on 2002/03/19 by llefebvr@llefebvre_laptop_r400_emu

removed debug traces

Change 19617 on 2002/03/19 by llefebvr@llefebvre_laptop_r400_emu

Corrected a bug in the SX that was causing the last quad of the triangle to be sent twice to the Rb. Also added debug traces in the SX.

Change 19600 on 2002/03/19 by sallen@sallen_r400_sun_marlboro

ferret shader pipe testbench work

Change 19449 on 2002/03/18 by llefebvr@llefebvre_laptop_r400_emu

Added RC_SC to chip.cpp...

Change 19434 on 2002/03/18 by jhoule@jhoule_r400_win_marlboro

Removed hard-coded texture addresses.
Changed alignment multiply from 6 to 8 in SQ.

Change 18970 on 2002/03/14 by llefebvr@llefebvre_laptop_r400_emu

Added the new SC_SX interface. Added SX_RB_quad interface. Added RB_SX interface. Modified SX_RB to reflect the HW interface.

Change 18930 on 2002/03/14 by sallen@sallen_r400_sun_marlboro

add export tweak for shader testing

Change 18463 on 2002/03/11 by sallen@sallen_r400_sun_marlboro

timing tweaks for ferret / shader pipe

Change 18443 on 2002/03/11 by llefebvr@llefebvre_laptop_r400_emu

Added shader type to the SP->SX data interface to simplify the life of ferret

Change 18345 on 2002/03/11 by llefebvr@llefebvre_laptop_r400_emu

Fixed a reservation station management problem in the sequencer related to pixel shader stalls.

Change 18337 on 2002/03/11 by llefebvr@llefebvre_laptop_r400_emu

Added export buffers into the SX block. Splitted the interface from the SQ/SP to the SX into 2 interfaces (data, control). Repaired compilation problems of ferret because of namespace missuses. Removed the VS_EXPORT_MASK register (replaced by a value of 0 in COUNTx registers).

Change 18326 on 2002/03/11 by sallen@sallen_r400_sun_marlboro

ferret shader pipe drive interp data in 512 chunks

Change 18033 on 2002/03/07 by sallen@sallen_r400_sun_marlboro

ferret update for work with shader pipe

Change 18029 on 2002/03/07 by askende@andi_crayola_emu_w

updated the ALU instruction opcode enumerated type to match the latest shader spec

Change 18028 on 2002/03/07 by askende@andi_crayola_emu_w

updated the opcode enumerated types to match the latest shader spec

Change 17825 on 2002/03/07 by sallen@sallen_r400_sun_marlboro

ferret - new interface routines
- debugging
- SQ_SP drives only pixel data

Change 17818 on 2002/03/07 by llefebvr@llefebvre_laptop_r400_emu

Changed the alu instruction packing so that it is consistant with what was done in the texture instruction packing and with the Shader pipe spec (byte 11 = MSB, byte 0 = LSB).

Change 17757 on 2002/03/06 by llefebvr@llefebvre_laptop_r400_emu

Added code to support the Control Flow instruction Execute. This code is not activated yet.

Change 17676 on 2002/03/06 by sallen@sallen_r400_sun_marlboro

ferret updates - emulator building .pipe files

Change 17543 on 2002/03/05 by llefebvr@llefebvre_laptop_r400_emu

Changed registers writes to be uint32 instead of uint8 for texture state

Change 17503 on 2002/03/05 by llefebvr@llefebvre_laptop_r400_emu

Another try to solve the endianess problem

Change 17498 on 2002/03/05 by llefebvr@llefebvre_laptop_r400_emu

endian check for unix...

Change 17398 on 2002/03/04 by sallen@sallen_r400_sun_marlboro

re-add ferret interface class needed

Change 17385 on 2002/03/04 by sallen@sallen_r400_sun_marlboro

ferret update for new interface driven pipe model

Change 17049 on 2002/02/28 by sallen@sallen_r400_sun_marlboro

clean up places ferret was hacked out
add 1st stage of interface based file/pipe in ferret
fix a few make file gotchas
fix a makefile out of order build that prevented a clean build from working

Change 16800 on 2002/02/27 by hartogs@fl_hartogs

Split VGT into its own block.

Change 16798 on 2002/02/27 by sallen@sallen_r400_sun_marlboro

ferret interfaces driven from run.cpp now
change Interface base class to drive ferret data

Change 16466 on 2002/02/25 by jhoule@jhoule_r400_win_marlboro

New 1.30 instruction.
Started integrating common instr/const class shared between TP and PrimLib.

Change 16385 on 2002/02/22 by vliu@vliu_r400_cnvliu100_win_cvd

Added support for BUILD_CONFIG environment variable.

Change 16305 on 2002/02/22 by jhoule@jhoule_r400_win_marlboro

Changed TPConst and TPInstr to have:
- pack and unpack instead of writeTo and readFrom
- packing to vector<uint32>&
- DWORD granularity function calls

Change 16176 on 2002/02/21 by jhoule@jhoule_r400_win_marlboro

Changed tp_const and tp_instr to be in a library in cmn_lib/src.
That way, code can be shared between emulator and primlib.

Change 15645 on 2002/02/15 by jhoule@jhoule_r400_win_marlboro

Removed relative paths for numbers.h (includes Charlton Wang's changelist)

Change 15436 on 2002/02/14 by llefebvr@llefebvre_laptop_r400_emu

Corrected a small bug in position export.

Change 15382 on 2002/02/13 by llefebvr@llefebvre_laptop_r400_emu

Fixed an error in the constant load of the SQ.

Change 15305 on 2002/02/12 by llefebvr@llefebvre_laptop_r400_emu

Added a count++ in handle_register_fn so that the constant data is written to sq memory...

Change 15275 on 2002/02/12 by llefebvr@llefebvre_laptop_r400_emu

Added valid bits between SQ and TP

Change 15268 on 2002/02/12 by llefebvr@llefebvre_laptop_r400_emu

Write the vertex data as bytes in the memory

Change 15243 on 2002/02/12 by sallen@sallen_r400_sun_marlboro

add ferret interface class with just one shader pipe

Change 15221 on 2002/02/12 by llefebvr@llefebvre_laptop_r400_emu

Updated SQ to take texture write masks into account

Change 15219 on 2002/02/12 by llefebvr@llefebvre_laptop_r400_emu

Updated TP->SQ interface to add write masking capabilities

Change 15218 on 2002/02/12 by llefebvr@llefebvre_laptop_r400_emu

Corrected an error in texture state management

Change 15217 on 2002/02/12 by llefebvr@llefebvre_laptop_r400_emu

The SX now only exports the valid positions to the VGT.

Change 15177 on 2002/02/11 by llefebvr@llefebvre_laptop_r400_emu

Interface sync problem on the PA->SQ vertex interface corrected

Change 15149 on 2002/02/11 by ygiang@ygiang_r400_win_marlboro

Added: Ferret hooks to shader Pipe

Change 15072 on 2002/02/11 by llefebvr@llefebvre_laptop_r400_emu

Corrected a memory alignement problem that corrupted the state of the arbiter. Also modified simple_triangle to load both the pixel and the vertex shaders.

Change 14992 on 2002/02/08 by llefebvr@llefebvre_laptop_r400_emu

There was an error in the vertex input function of the sequencer it is now fixed and you can load a partial group of vertex.

Change 14979 on 2002/02/08 by llefebvr@llefebvre_laptop_r400_emu

Cheking in the last modifications of the SQ SX before trying to integrate the PA block.

Change 14941 on 2002/02/08 by jhoule@jhoule_r400_win_marlboro

Was calling wrong function for filling TPInstr.

Change 14939 on 2002/02/08 by jhoule@jhoule_r400_win_marlboro

iostream.h --> iostream

Change 14791 on 2002/02/06 by llefebvr@llefebvre_laptop_r400_emu

Removed all sequencer related texture pipe hardcoded portions of the texture request.

Change 14783 on 2002/02/06 by llefebvr@llefebvre_laptop_r400_emu

Fixed the order in which the instructions are written to memory.

Change 14721 on 2002/02/06 by llefebvr@llefebvre_laptop_r400_emu

Updated the SQ->SX interface to reflect that the ParameterCaches are now in the SX block.

Change 14627 on 2002/02/05 by llefebvr@llefebvre_laptop_r400_emu

Created vertex.sp and pixel.sp. Simplified the shaders to their simplest expression for the 2/22 milestone. Updated the golden images to reflect the change.

Change 14530 on 2002/02/04 by llefebvr@llefebvre_laptop_r400_emu

Not seeing the correct data when using IM_LOADS...

Change 14381 on 2002/02/01 by hwise@fl_hwise_r400_win

1) Moved RBBM_CNTL and RBBM_SOFT_RESET primary register
  aperture addresses to match the I/O addresses
2) Removed RBBM_*_GO_ASSERT registers
3) Added handleRegisterAccess() functions to PA and SQ
  block classes to catch register read/write access
  broadcast by RBBM (stub with small decode example)
4) More changes to PM4 decode of IM_LOAD and SET_CONSTANT
5) Removed a few PM4 type3 packet definitions from primlib
  that will not be added to R400 as was previously thought
6) Updated primlib pm4lib.cpp Promo4Lib constructor to

  force the disabling of CP microcode

Change 14318 on 2002/01/31 by llefebvr@llefebvre_laptop_r400_emu

Removed all temporary registers from the SQ (but for the _CNT registers wich are needed until we implement CF). Primilib now needs to send the correct PM4 packets to the CP in order to adjust the SQ registers before sending the first primitive.

Change 14215 on 2002/01/30 by llefebvr@llefebvre_laptop_r400_emu

Added multiple triangle support in the interpolators. Also added PC allocation scheme. Also added the SP->Parameter cache dummy interface.

Change 13962 on 2002/01/28 by rbeaudin@rbeaudin_r400_win_marlboro

start of making the transaction engine working

Change 13934 on 2002/01/25 by llefebvr@llefebvre_laptop_r400_emu

Added fields to the SQ_SP_Interp interface.

Change 13924 on 2002/01/25 by llefebvr@llefebvre_laptop_r400_emu

Added two SQ->SP dummy interfaces to help with HW block level testing.

Change 13871 on 2002/01/25 by jhoule@jhoule_r400_win_marlboro

Now uses standard header <iostream>.

Change 13829 on 2002/01/25 by llefebvr@llefebvre_laptop_r400_emu

Added SX_PA interface fixed remaining bugs regarding register integration.

Change 13785 on 2002/01/24 by llefebvr@llefebvre_laptop_r400_emu

Fixed the registers but the writes do not go thru because of a primilib problem?

Change 13725 on 2002/01/24 by hwise@fl_hwise_r400_win

Integrated "play area" register spec into the emulator tree

Change 13669 on 2002/01/23 by llefebvr@llefebvre_laptop_r400_emu

Added SX->SQ interface. Updated the Arbiter to take into account co-issue exporting restrictions...

Change 13592 on 2002/01/23 by llefebvr@llefebvre_laptop_r400_emu

Added File headers to all SQ and SX files.

Change 13589 on 2002/01/23 by llefebvr@llefebvre_laptop_r400_emu

Added the SX block to the emulator.

Change 13450 on 2002/01/21 by llefebvr@llefebvre_laptop_r400_emu

Added the Idle function. Updated sanity. Won't regress because of framebuffer0. Ray is working on it...

Change 13387 on 2002/01/21 by llefebvr@llefebvre_laptop_r400_emu

Added Vertex functionnalities to the SQ. Remaining interfaces to be added are SQ->SX position
and SX->PA position.

Change 13288 on 2002/01/18 by llefebvr@llefebvre_laptop_r400_emu

Added Texture pipe functionnality to the SQ.

Change 13153 on 2002/01/16 by llefebvr@llefebvre_laptop_r400_emu

Added the PA->SQ vertex interface

Change 13149 on 2002/01/16 by llefebvr@llefebvre_laptop_r400_emu

Refined the interfaces

Change 12931 on 2002/01/11 by llefebvr@llefebvre_laptop_r400_emu

added the GPR allocation for the static mode only for both Vertices and pixel.

Change 12791 on 2002/01/10 by llefebvr@llefebvre_laptop_r400_emu

Fixed an interpolation problem when using more than 2 interpolated parameters

Change 12782 on 2002/01/10 by llefebvr@llefebvre_laptop_r400_emu

Fix for Jocelyn to work on the TP. This compiles and runs but gives incorect results for now.

Change 12750 on 2002/01/09 by llefebvr@llefebvre_laptop_r400_emu

Added TSTATE_MEM_BASE_ADDR register and TSTATE_MEM_WORD_COUNT registers in order to be able to load the texture state from memory. also added a dllexported function in the SQ to to this. Added a texture state store to the SQ.

Change 12674 on 2002/01/09 by llefebvr@llefebvre_laptop_r400_emu

Added a condition in the arbiter that prevents groups of pixel/vertices from passing each other

Change 12633 on 2002/01/09 by llefebvr@llefebvre_laptop_r400_emu

There was a color channel alignement problem in the RB. IT is now fixed but the golden image have to be regenerated yet again. It should be in tones of red and yellow (NOT blue and green).

Change 12617 on 2002/01/08 by askende@andi_crayola_emu_w

rearranged the order of the channels (abgr or wzyx) ......and completed the mas k capability in the GPR write back logic.

Change 12613 on 2002/01/08 by llefebvr@llefebvre_laptop_r400_emu

channel Write Mask instruction read error in the SQ corrected

Change 12609 on 2002/01/08 by llefebvr@llefebvre_laptop_r400_emu

Changed the channel order to RGBA (R in LSB A in MSB). Emulator wise it means that R is in field 0 and A in field 3.

Change 12605 on 2002/01/08 by llefebvr@llefebvre_laptop_r400_emu

Intermediate changes compiles and runs but doesn't give the right answer

Change 12577 on 2002/01/08 by llefebvr@llefebvre_laptop_r400_emu

Added sources as const variables instead ot regular.

Change 12565 on 2002/01/07 by askende@andi_crayola_emu_w

1.added support for argument selection being a constant

2.added masking capabilities for the write bacl into GPRs

Change 12554 on 2002/01/07 by askende@andi_crayola_emu_w

checking in for backup purposes

Change 12500 on 2002/01/07 by llefebvr@llefebvre_laptop_r400_emu

reduced the size of the SQ_TP interface and added some comments in arbiter.cpp

Change 12441 on 2002/01/04 by askende@andi_crayola_emu_w

Backup do not sync to this changelist it doesn't compile

Change 12388 on 2002/01/04 by llefebvr@llefebvre_laptop_r400_emu

interleaving ALU clauses

Change 12342 on 2002/01/03 by llefebvr@llefebvre_laptop_r400_emu

added ready signal between the PA and the SQ

Change 12328 on 2002/01/03 by llefebvr@llefebvre_laptop_r400_emu

added scalar support in the sequencer. No scalar exports yet.

Change 12177 on 2001/12/21 by llefebvr@llefebvre_laptop_r400_emu

major changes in the SQ block. Now supports clauses. Not fully tested but otherwise operationnal.

Change 12084 on 2001/12/20 by llefebvr@llefebvre_laptop_r400_emu

sbfloats instead of sfloats

Change 12064 on 2001/12/19 by llefebvr@llefebvre_laptop_r400_emu

New interface definition between SQ and TP

Change 12060 on 2001/12/19 by llefebvr@llefebvre_laptop_r400_emu

constants are now working.

Change 12008 on 2001/12/19 by llefebvr@llefebvre_laptop_r400_emu

Changed constants order. Added Constant read support, compiles and runs but doesn't work.

Change 11940 on 2001/12/18 by llefebvr@llefebvre_laptop_r400_emu

repaired a problem with RB valid bits. Added constant support, need to load them using the MC.

Change 11912 on 2001/12/18 by llefebvr@llefebvre_laptop_r400_emu

new ALU instruction format and clamp instructions now in the performance emulator. First golden image can now be generated.

Change 11764 on 2001/12/14 by askende@andi_crayola_emu_w

rearranged the ALU instruction word definition

added the clamping for the ALU results

Change 11581 on 2001/12/12 by rbeaudin@rbeaudin_r400_win_marlboro

changing emu_lib structure

```
Change 173774 on 2004/06/15 by donaldl@donaldl_xenos_linux_orl

        1.  Changed DEBUSSY_PATH to VERDI_ROOT in buildtb_gate.
        2.  Updated sq to sp trackers to not compare the shader pipe defined by
            ROM_SIMD_SEL[1:0] and ROM_PIPE_SEL[3:0] if rsp is enabled.

Change 170775 on 2004/05/28 by bhankins@bhankins_xenos_linux_orl

        back integrate sx from xenos

Change 169691 on 2004/05/24 by rramsey@rramsey_xenos_linux_orl

        integrate xenos changes back to r400 for pa and vgt
        I verfied milestone_tri passed tb_pa and vgt_tb, but did not do a full
        block level regression. release_parts_lib passed.

Change 169281 on 2004/05/21 by rramsey@rramsey_xenos_linux_orl

        integrate xenos changes to r400 for sq, sp/rsp, spi
        take top-level-registers to r400, but only sc_cp_tlr0 is instanced as an example

Change 158310 on 2004/03/29 by dclifton@dclifton_r400

        Updates for ram and bist changes

Change 154269 on 2004/03/11 by rramsey@rramsey_xenos_linux_orl

        integrate fix for constant store hang

Change 153809 on 2004/03/09 by llefebvr@llefebvr_r400_linux_marlboro

        Integration of the Loop rep fix.

Change 153542 on 2004/03/08 by donaldl@donaldl_xenos_linux_orl

        Qualified RSP comparing of data with sq_vc_fetch_type to fix erroneous mismatches.

Change 153403 on 2004/03/08 by mearl@mearl_r400_linux_orl

        Took out LOD Correction from SC, SC/SQ interface, SQ, SQ/TP interface.

Change 153372 on 2004/03/08 by vromaker@vromaker_r400_linux_marlboro

        - hooked u0, u1, u2, and u3_TP_SP_data_valid signals up to thier respective shader pipes
          (instead of the version that ORed all the valids together)

Change 153086 on 2004/03/05 by danh@danh_xenos_linux_orl

        removed LOD changes
```

```
Change 153085 on 2004/03/05 by danh@danh_xenos_linux_orl

        Correct Virage 90nm changes and SX gate level simulation changes

Change 153082 on 2004/03/05 by danh@danh_r400_win

        Virage 90 nm changes and SX gate level simulation changes

Change 152936 on 2004/03/05 by danh@danh_r400_win

        integrated from //depot/xenos

Change 152848 on 2004/03/05 by mmantor@mmantor_xenos_linux_test

        <fixed a bug in the loading of aluconst, back integrated removal of realtime space from
aluconst and texconst mems and control logic in rbi and all hookups, altered sq_regress per
carlos test set>

Change 152426 on 2004/03/03 by vromaker@vromaker_r400_linux_marlboro

        - bug fix for AIS update state machine: need to look at both unregistered
          and registered updates in state 0

Change 152169 on 2004/03/02 by vromaker@vromaker_r400_linux_marlboro

        - added a signal to enable/disable early thread buffer updates (it's
          called enable_early_update and it's tied high)

Change 152105 on 2004/03/02 by rramsey@rramsey_xenos_linux_orl

        Fix loop index relative addressing for negative indices and error cases

Change 151675 on 2004/02/27 by vromaker@vromaker_r400_linux_marlboro

        - fix for AIS update state machine

Change 151644 on 2004/02/27 by vromaker@vromaker_r400_linux_marlboro

        - fix for random 82d failure: reset ppb_exec_cnt when new CFI is passed to exec state
machine (was using an old
          exec count in a case where the pred condition failed)
        - removed a RT test from the sq mini regress list

Change 151466 on 2004/02/27 by danh@danh_xenos_linux_orl

        backed out change #168

Change 151457 on 2004/02/27 by danh@danh_r400_win
```

```
        integration from //depot/xenos

Change 151426 on 2004/02/26 by donaldl@donaldl_xenos_linux_orl

        Integrated from xenos:
        1) Took out RT stream logic in sc_packer
        2) Integrated SC_B back with SC
        3) Updated files to accomodate GATE level sims

Change 151419 on 2004/02/26 by danh@danh_r400_win

        integrated from //depot/xenos

Change 151418 on 2004/02/26 by danh@danh_r400_win

        integrated from //depot/xenos

Change 151366 on 2004/02/26 by danh@danh_r400_win

        integrated from //depot/xenos

Change 151318 on 2004/02/26 by rramsey@rramsey_xenos_linux_orl

        Fix exec machine so it doesn't change resource or serialize bits when
        sending back non-exec instr (pred jump, loop_end, etc)

Change 151211 on 2004/02/26 by bhankins@bhankins_xenos_linux_orl

        Add support for generating event quads from sx to bc/rb.
           Support is disabled for now in src/common/sx_defines.v
           This code supports behavioral memory only for now.

Change 151177 on 2004/02/25 by rramsey@rramsey_xenos_linux_orl

        Change cfs eject to use bit from cf instr, and fix bug with ejection
        of a partially executed instr.
        Change status reg and vtx_ctl to use vgt define for fetch_done

Change 151101 on 2004/02/25 by donaldl@donaldl_xenos_linux_orl

        Removed real-time stream reg mems in SX and removed the i/o signal
        SQ_SX_rt_sel between the SQ and SX.

Change 151067 on 2004/02/25 by vromaker@vromaker_r400_linux_marlboro

        - waterfall fix for pipelined AIS thread buffer update (found by random 80e)

Change 150962 on 2004/02/25 by llefebvr@llefebvr_r400_linux_marlboro
```

```
        This is fixing the RS_FULL and GPR_STALL performance counters.

Change 150374 on 2004/02/21 by mmantor@mmantor_xenos_linux_test

        <fixed a synthesis error and enabled the 256 deep export buffers>

Change 150220 on 2004/02/20 by rramsey@rramsey_xenos_linux_orl

        replace the old vtx/pix tex cf trackers with a new unified tracker
        add vc cf tracker
        move some r400-only sx signals inside an ifdef
        delete obsolete trackers

Change 150184 on 2004/02/20 by llefebvr@llefebvr_r400_linux_marlboro

        Adding another minor fix on the latency counters.

Change 150005 on 2004/02/19 by vromaker@vromaker_r400_linux_marlboro

        - fixed a bug in the pipelined thread buffer update that was causing a failure in random
testing

Change 149787 on 2004/02/18 by rramsey@rramsey_xenos_linux_orl

        Add register bits for disabling arb and cfs eject.
        Add clause eject capability to cfsm and logic to the thread arb
        to drive the eject.
        Remove a stage from the cfs ppb and the isr from the tif in order
        to reduce the amount of work that can be ahead of the next best thread.
        Fix a bug with SQ_SP_fetch_swizzle mux between tp and vc values.
        Swap phasing of fetch and alu instruction fetch is reads to match up
        better with when the cfs can deliver an instr.
        Update control flow tracker to be able to handle clause ejection.
        Fix some compile warnings in tb_sqsp.

Change 149566 on 2004/02/17 by rramsey@rramsey_xenos_linux_orl

        fix some bad logic that synopsys was complaining about

Change 149537 on 2004/02/17 by jhoule@jhoule_r400_linux_marlboro

        tp_sqsp.dmp tracker update

        TP_SQSP_Dump:
        - Changed serialization to be exactly what the RTL spits instead of the bastardized 4x32
cycling; controlled with Use_New_TP_SQSP_Dump environment variable)
        - Added RSP data per pipe (only available when using new serialization)
```

- Added rf_expand_enable ***RIGHT AFTER xyzw_parity*** in order to get packed formats to work properly

TexturePipe:
- Added functions to prepare RSP data for the TP_SQSP_Dump

Testbench:
- Added rf_expand_enable *JUST AFTER* xyzw_parity (this affects the old path); was tied to 0
- Connected data_format instead of tying it to 6'h26 (FMT_32_32_32_32_FLOAT)

FormatOracle:
- Added column representing the encoded format sent to the formatter
- Kept weird issue where DXN (and other formats) have 2x16 channels in TP instead of 2x8(.8). Doesn't seem to be used anyways.

Change 149472 on 2004/02/17 by mmantor@mmantor_xenos_linux_test

<back out usage of 512 export buffer locations until problem debugged>

Change 149458 on 2004/02/17 by mmantor@mmantor_xenos_linux_orl

<the remainder of my previous check in that got left out by error>

Change 149382 on 2004/02/15 by mmantor@mmantor_xenos_linux_orl

<added counter to sq for flow control of sx alloc table, hook up state control for depth of sx buffers and set the default depth to 256 and initial hook up of event quads for sq and sx>

Change 149262 on 2004/02/13 by llefebvr@llefebvr_r400_linux_marlboro

Adding Event performance counters.

Change 149131 on 2004/02/13 by vromaker@vromaker_r400_linux_marlboro

- fix for AIS update to take waterfalling into account
- only change to CFS was to assign some signals to logic expressions

Change 149113 on 2004/02/13 by llefebvr@llefebvr_r400_linux_marlboro

Fixing latency counters. They were incorectly counting event threads.

Change 149058 on 2004/02/13 by donaldl@donaldl_xenos_linux_orl

Bug fix: delay no_compare flags to line up with SP to SX data.

Change 148947 on 2004/02/12 by rramsey@rramsey_xenos_linux_orl

change a loop int to a unique name for synthesis

Change 148704 on 2004/02/11 by vromaker@vromaker_r400_linux_marlboro

- signal name change only in CFS (changed cfi_no_pred to cfi_pred_clean)
- added reset for pc_base and export_id fields of status register

Change 148410 on 2004/02/10 by vromaker@vromaker_r400_linux_marlboro

- change to handle concurrent updates from all 4 AIS's that go to an ALU State Memory
- the update info from each AIS is registered, and a state machine sequences the writes

Change 148380 on 2004/02/10 by bhankins@bhankins_xenos_linux_orl

Send simd_id to sx from sq one clock earlier, then register and break up selects to redundancy muxes in sx.

Change 148246 on 2004/02/09 by donaldl@donaldl_xenos_linux_orl

Added flat_shading signal for use in the SX parameter subtract function.
(ie. if flat_shading is true, ignore infinity checks; just do subtract. Result
should be zero.)

Change 148140 on 2004/02/09 by rramsey@rramsey_xenos_linux_orl

Changes to allow the thread arbiters to pick the best thread every four
clocks, rather than picking one and holding it until the cf machine takes
it.
Also breaks ties between export_arb and thread_arbs by having the space
allocs happen as a separate process not requiring the thread_arbs.

Change 147861 on 2004/02/06 by vromaker@vromaker_r400_linux_marlboro

- fix for non-delayed AIS update of thread buffer
- required that all ais update inputs to the thread buffer be qualified with thread type
  before being used

Change 147453 on 2004/02/05 by donaldl@donaldl_xenos_linux_orl

Allow disabling of clocks to each of the vsp's in the SP. Disabling clocks is
done either when redundancy is used or when disabling of a simd pipe.

Change 147418 on 2004/02/05 by mearl@mearl_r400_win

update status

Change 147410 on 2004/02/05 by amys@amys_xenos_lnxrgs_orl

added path for xenos path for fullchip, so trackers won't break xenos build every time an integration is done

Change 147378 on 2004/02/04 by danh@danh_r400_win

status update

Change 147377 on 2004/02/04 by danh@danh_xenos_linux_orl

Changed type1_export_size generation, now when vs_export_mode = 0 or 7 it will be a 1 position export.

Change 147106 on 2004/02/03 by danh@danh_r400_win

status update

Change 147059 on 2004/02/03 by danh@danh_r400_win

status update

Change 147047 on 2004/02/03 by mearl@mearl_r400_win

update status

Change 147045 on 2004/02/03 by danh@danh_r400_win

status update

Change 146966 on 2004/02/03 by mearl@mearl_r400_win

update status

Change 146945 on 2004/02/03 by danh@danh_r400_win

status update

Change 146923 on 2004/02/02 by danh@danh_r400_win

status update

Change 146705 on 2004/02/01 by mmantor@mmantor_xenos_linux_orl

<moved a register from sq to sp for tc and vc fetch address so the min latency through sp is 3 clocks in prep for intrinsity and moved extra register to back of sp for either future remove or use for top level routing and sent export simd sel 2 clocks later once for minimal latency and other for register movement. also changed trackers and mvoed register in rsp>

Change 146511 on 2004/01/30 by vromaker@vromaker_r400_linux_marlboro

- AIS now asserts ais_done back to the thread buffer without any delays

- AIS now asserts ais_update back to the thread buffer without any delays if the last instruction
  of a clause is not a pred_set or a kill (if it is, the update is delayed until the SP data is
  written back to the SQ)

Change 146497 on 2004/01/30 by danh@danh_r400_win

status update

Change 146431 on 2004/01/30 by danh@danh_r400_win

status update

Change 146361 on 2004/01/30 by vromaker@vromaker_r400_linux_marlboro

picked two tests

Change 146353 on 2004/01/30 by danh@danh_r400_win

status update

Change 146329 on 2004/01/30 by danh@danh_r400_win

status update

Change 146132 on 2004/01/29 by danh@danh_r400_win

status update

Change 146066 on 2004/01/29 by rramsey@RRAMSEY_P4_r400_win

get rid of some invalid testcases

Change 145994 on 2004/01/28 by danh@danh_r400_win

status update

Change 145934 on 2004/01/28 by danh@danh_r400_win

status update

Change 145919 on 2004/01/28 by mearl@mearl_r400_win

update status

Change 145847 on 2004/01/28 by rramsey@RRAMSEY_P4_r400_win

update with 1/28 regression results

```
Change 145615 on 2004/01/27 by bhankins@bhankins_xenos_win_orl

     update status

Change 145611 on 2004/01/27 by smoss@smoss_crayola_linux_orl_regress

       new path

Change 145568 on 2004/01/27 by vromaker@vromaker_r400_linux_marlboro

     status update on r400sq_flow_control_rts_16 and _19

Change 145549 on 2004/01/27 by danh@danh_r400_win

     status update, picked new test

Change 145535 on 2004/01/27 by danh@danh_xenos_linux_orl

     |tx_instr[4:0] is now used for the mux select of tfetch_swapped_bit

Change 145533 on 2004/01/27 by danh@danh_xenos_linux_orl

     |tx_instr[4:0] is now used for the mux select of tfetch_swapped_bit

Change 145355 on 2004/01/26 by llefebvr@llefebvr_r400_linux_marlboro

     Added 4 performance counters to the SQ to measure the latency of pixel and vertex
threads from the time they enter te Sq to the time they leave.

Change 145135 on 2004/01/23 by rramsey@rramsey_xenos_linux_orl

     Update yield_optimize test so it tests the logic better
     Fix cfsm clause boundary detection for yield_optimize and add setting
     of exsm_updating to EX_EXEC state when it is going to update

Change 145037 on 2004/01/23 by mearl@mearl_r400_win

     update status

Change 144945 on 2004/01/23 by ctaylor@ctaylor_xenos_linux_orl

      Fixed tracker bug for
NOP,RETURN,LOOP_START,LOOP_END,COND_CALL,COND_JUMP and ALLOC where
vc_request field was left out of tracker compare so all subsequent fields came from the wrong
column in the dump file.

Change 144878 on 2004/01/23 by rramsey@RRAMSEY_P4_r400_win

     update with 1/22 regression results
```

```
Change 144816 on 2004/01/22 by smoss@smoss_xenos_linux_orl

       correct brain dump

Change 144711 on 2004/01/22 by rramsey@rramsey_xenos_linux_orl

     Add vs fetch done to SQ
     Fix testbench handling of vizq events again

Change 144679 on 2004/01/22 by mearl@mearl_r400_win

     update status

Change 144656 on 2004/01/22 by vromaker@vromaker_r400_linux_marlboro

     updated status and picked a test

Change 144611 on 2004/01/22 by llefebvr@llefebvr_r400_emu_montreal

     Update status.

Change 144609 on 2004/01/22 by llefebvr@llefebvr_r400_linux_marlboro

     Putting back the fix of Dan Harmon for back to back exports inadvertly reverted by Mike
M.

Change 144482 on 2004/01/22 by llefebvr@llefebvr_r400_emu_montreal

     picking up some tests.

Change 144466 on 2004/01/22 by smoss@smoss_crayola_linux_orl_regress

       updated test.cfg for some missing dumps

Change 144373 on 2004/01/21 by danh@danh_r400_win

     picked a new test

Change 144326 on 2004/01/21 by rramsey@RRAMSEY_P4_r400_win

     update status with 1/21 results

Change 144186 on 2004/01/21 by danh@danh_xenos_linux_orl

     fixed ps_const_max typo

Change 144030 on 2004/01/20 by llefebvr@llefebvr_r400_emu_montreal
```

```
     update status for kill tests.

Change 144029 on 2004/01/20 by rramsey@rramsey_xenos_linux_orl

     clean up ctl_flow_seq
     add ais_thread_type to ais_update snoop in cfs

Change 143983 on 2004/01/20 by vromaker@vromaker_r400_linux_marlboro

     - small update to load and hold the vtx and pix constant base registers the same
       way as the input staging register

Change 143952 on 2004/01/20 by mearl@mearl_xenos_linux_orl

     1. Added EOP to sc_pix_vec_grp_out.dmp file to keep track of RT streams for tb_sqsp
testbench
          sc_dumps.cpp
          sc_dumps.h
          sc_interp.cpp
          tb_sqsp.v
        2. Fixed bug; when SIMD w/ 0 bad pipes is followed by SIMD w/ 2 bad pipes,
wrong SP is selected.
          sq_vtx_ctl.v
        3. Made timing fixes.
          sc_packer.v

Change 143929 on 2004/01/20 by danh@danh_r400_win

     status update

Change 143883 on 2004/01/20 by vromaker@vromaker_r400_linux_marlboro

     status update, picked new test

Change 143755 on 2004/01/19 by vromaker@vromaker_r400_linux_marlboro

     status update, picked another test

Change 143726 on 2004/01/19 by bhankins@bhankins_xenos_linux_orl

     Mods to support tb_sqsp testing with xenos

Change 143618 on 2004/01/18 by rramsey@rramsey_xenos_linux_orl

     Rework ctl flow sequencer so it can start a new thread every 4 clocks

Change 143377 on 2004/01/16 by danh@danh_r400_win

     status update
```

```
Change 143312 on 2004/01/15 by vromaker@vromaker_r400_linux_marlboro

     - removed the SQ_SP thread ID and type ports from the SQ; they were
       conditionally compiled 'ifdef SIM, but that was not working for
       xenos
     - the trackers now reference these signals at the SQ level

Change 143243 on 2004/01/15 by vromaker@vromaker_r400_linux_marlboro

     - fixed input to AIQ FIFO by adding an input staging register and
       loading it on valid transfers from the instr fetcher

Change 143160 on 2004/01/15 by vromaker@vromaker_r400_linux_marlboro

     - fixes for unconnected ports
     - had to make some changes to tb_sqsp and sim.cfg based on the removal of
       SQ_SP_instruct_start
     - also qualified predicate and valid bit writes with waterfall mask in ais_output

Change 143119 on 2004/01/14 by danh@danh_r400_win

     status update

Change 142926 on 2004/01/14 by rramsey@rramsey_xenos_linux_orl

     change tracker to look at sq port rather than connecting signal

Change 142689 on 2004/01/13 by danh@danh_r400_win

     status update

Change 142672 on 2004/01/13 by dclifton@dclifton_r400

     Update for changes in sp.

Change 142573 on 2004/01/13 by danh@danh_xenos_linux_orl

     SQ_SP_vsr_rd_addr is now only compared when SQ_SP_gpr_input_mux = 2 (VSRs
selected)

Change 142569 on 2004/01/13 by bhankins@bhankins_xenos_win_orl

     update status

Change 142543 on 2004/01/13 by bhankins@bhankins_xenos_win_orl

     update
```

Change 142536 on 2004/01/13 by bhankins@bhankins_xenos_win_orl

pick test

Change 142533 on 2004/01/13 by bhankins@bhankins_xenos_win_orl

update status

Change 142219 on 2004/01/12 by rramsey@RRAMSEY_P4_r400_win

update with 1/11 regression results

Change 142212 on 2004/01/12 by mmantor@mmantor_xenos_linux_orl

<this is val's change for a timing fix in the pa and Vic's changes for the sq which include coding of special flow control optimizations and some timing fixes for the sq>

Change 142124 on 2004/01/10 by danh@danh_r400_win

Status update.

Change 141976 on 2004/01/09 by danh@danh_r400_win

Status update.

Change 141969 on 2004/01/09 by danh@danh_xenos_linux_orl

aiq_instr[15] (export bit) now forces prev_vector_mask_q and prev_scalar_mask_q to 0, this allows back to back export instructions to work properly.

Change 141964 on 2004/01/09 by danh@danh_xenos_linux_orl

gen_index_cycle now forces a count_match[3:0]

Change 141926 on 2004/01/09 by mearl@mearl_r400_win

update status

Change 141921 on 2004/01/09 by rramsey@RRAMSEY_P4_r400_win

updating status

Change 141806 on 2004/01/09 by amys@amys_xenos_linux_orl

modified read-back of sq_flow_control_reg by inserting a bit so that the read-back matches the spec

Change 141650 on 2004/01/08 by rramsey@rramsey_xenos_linux_orl

Add SQ_CP_vs_fetch_done to sq, tied low for now
Fix the way vizq_start events are handled in tb_sqsp

Change 141418 on 2004/01/08 by vromaker@vromaker_r400_linux_marlboro

updated status - picked tests

Change 141401 on 2004/01/08 by rramsey@RRAMSEY_P4_r400_win

update with 1/8 results

Change 141385 on 2004/01/08 by bhankins@bhankins_xenos_linux_orl

Initial add of thread done/event logic in sx

Change 141155 on 2004/01/07 by rramsey@rramsey_xenos_linux_orl

Fix a bug that was sending writes to the wrong phys addr if a new
pa was allocated before the write actually happened

Change 140800 on 2004/01/06 by rramsey@RRAMSEY_P4_r400_win

update with 1/6 status, pick a test

Change 140782 on 2004/01/06 by mmantor@mmantor_xenos_linux_orl

<more timing fixes>

Change 140701 on 2004/01/05 by rramsey@rramsey_xenos_linux_orl

add an sc state push for the vs_done event

Change 140615 on 2004/01/05 by vromaker@vromaker_r400_linux_marlboro

updated status, picked new tests

Change 140594 on 2004/01/05 by rramsey@RRAMSEY_P4_r400_win

update with 1/1 regression results

Change 140576 on 2004/01/05 by danh@danh_r400_win

Status update.

Change 140556 on 2004/01/05 by smoss@smoss_crayola_win

update with latest status

Change 140496 on 2004/01/04 by mmantor@mmantor_xenos_linux_orl

<removed o_rbi_rd_data from the reset clk process>

Change 140451 on 2004/01/02 by mmantor@mmantor_xenos_linux_orl

<another timing fix>

Change 140441 on 2004/01/02 by mmantor@mmantor_xenos_linux_orl

<fixes for timing paths in the sq and pav>

Change 140380 on 2003/12/31 by vromaker@vromaker_r400_linux_marlboro

- fix for dropped real_time flag: moved export_pos bit into the "flags" field
  (it was using a bit that was set aside for the extra_in field)

Change 140350 on 2003/12/31 by rramsey@rramsey_xenos_linux_orl

Fix a bug that was allowing the texconst mem to be written when full (no phys addr available)

Change 140331 on 2003/12/31 by mearl@mearl_xenos_linux_orl

Fixed bug; was using thread type instead of fetch type.

Change 140313 on 2003/12/31 by jcarroll@jcarroll_r400_win

added latest status; picked new test

Change 140270 on 2003/12/30 by danh@danh_r400_win

Status update.

Change 140205 on 2003/12/30 by rramsey@RRAMSEY_P4_r400_win

update test status, pick another one

Change 140203 on 2003/12/30 by danh@danh_r400_win

Status update.

Change 140174 on 2003/12/30 by rramsey@RRAMSEY_P4_r400_win

update status

Change 140155 on 2003/12/30 by rramsey@RRAMSEY_P4_r400_win

update with 12/30 regression results

Change 140117 on 2003/12/30 by danh@danh_r400_win

Status update.

Change 140051 on 2003/12/29 by mearl@mearl_r400_win

update status

Change 140050 on 2003/12/29 by vromaker@vromaker_r400_linux_marlboro

status update - emulator fix was made for scalar const opcodes

Change 140036 on 2003/12/29 by jcarroll@jcarroll_r400_win

Updated jcarroll status

Change 139909 on 2003/12/28 by mmantor@mmantor_xenos_linux_orl

<timing fixes>

Change 139795 on 2003/12/23 by danh@danh_r400_win

Status update.

Change 139383 on 2003/12/23 by ctaylor@ctaylor_xenos_linux_orl

Fixed bug in control flow sequencer where when thread was put back onto thread buffer due to alloc cfi, the no-serialize bit was being taken from bit 40 of the cfs opcode instead of the execute state machine opcode so it was the right bit from the wrong instruction. Things have been working mostly due to the fact that bit 40 of most of the other CFI opcodes is reserved and therefore 0.

Change 139373 on 2003/12/23 by llefebvr@llefebvr_r400_emu_montreal

updated status for SX->PA missmatches.

Change 139338 on 2003/12/23 by vromaker@vromaker_r400_linux_marlboro

- fix for scalar const ops: y and x swizzle fields used for gpr address bits [5:4]
  and [3:2] were swapped

Change 139327 on 2003/12/23 by rramsey@rramsey_xenos_linux_orl

add simd_id to sp_out mismatch message
make sx_rb_color tracker multi-threaded per sx/rb interface

Change 139310 on 2003/12/23 by vromaker@vromaker_r400_linux_marlboro

status update - took another test

Change 139223 on 2003/12/22 by llefebvr@llefebvr_r400_emu_montreal

Now working on SX->PA missmatches.

Change 139193 on 2003/12/22 by rramsey@RRAMSEY_P4_r400_win

update status

Change 139142 on 2003/12/22 by llefebvr@llefebvr_r400_emu_montreal

The SX->RB tracker is bad. Added a comment to explain the situation.

Change 139097 on 2003/12/22 by ctaylor@ctaylor_xenos_linux_orl

Fixed bug related to clamping of GPR addresses which are out of range. Old code clamped to absolute zero instead of the base for the current thread.

Change 139066 on 2003/12/22 by bhankins@bhankins_xenos_linux_orl

Add tbtrk_sx_bc_quad tracker (C1 version only)

Change 139057 on 2003/12/22 by rramsey@RRAMSEY_P4_r400_win

update more tests, pick another one

Change 139050 on 2003/12/22 by rramsey@RRAMSEY_P4_r400_win

Update with new regression results, pick a test

Change 139045 on 2003/12/22 by rramsey@rramsey_xenos_linux_orl

Fix phasing of thread_count in sq_ais_output.
Fix o_gprsm_busy from sq_vtx_ctl and change thread counter to only
reset on RST_VTX_CNT event.

Change 138986 on 2003/12/20 by mmantor@mmantor_xenos_linux_orl

<changed csim to only make one pass for param gen and gen index and write the dump files correctly, fixed a timing loop in pix tthread buffer >

Change 138662 on 2003/12/19 by mearl@mearl_r400_win

update status

Change 138650 on 2003/12/19 by danh@danh_r400_win

Status update.

Change 138588 on 2003/12/19 by vromaker@vromaker_r400_linux_marlboro

- fixed a copy-paste error in the new code that generates src_c_sel for scalar const ops

Change 138586 on 2003/12/19 by mearl@mearl_r400_win

update status

Change 138491 on 2003/12/18 by llefebvr@llefebvr_r400_emu_montreal

I'll leave the _const_add test to Vic since he is working on it with Dan. I'll focus instead on coissue_frac_01.

Change 138489 on 2003/12/18 by mearl@mearl_r400_win

update status

Change 138486 on 2003/12/18 by danh@danh_r400_win

Status update.

Change 138455 on 2003/12/18 by mearl@mearl_r400_win

update status

Change 138309 on 2003/12/18 by danh@danh_r400_win

Status update.

Change 138289 on 2003/12/17 by rramsey@rramsey_xenos_linux_orl

Fix a bug with pred_override that can occur when a clause starts
with two predicated alu instructions. pred_override needs to use
the isr version of the pred bits in this case because the pred register
can't be intit'ed until after the the last instr of the prev clause has
a chance to return its pred values and push them back to the thread buffer.
This fixes r400sc_sp_sample_cntl_47 and hopefully many more.

Change 138288 on 2003/12/17 by vromaker@vromaker_r400_linux_marlboro

- fix for scalar const opcodes: src_c_sel and gpr_read_en logic was updated

Change 138212 on 2003/12/17 by mearl@mearl_r400_win

update status

Change 138152 on 2003/12/17 by danh@danh_r400_win

Status Update.

Change 138138 on 2003/12/17 by bhankins@bhankins_xenos_linux_orl

1. Modify detailed quad fifo to keep mrt quads and mem export quads together
2. Change NEW_BC defines to C1

Change 138087 on 2003/12/17 by rramsey@RRAMSEY_P4_r400_win

latest regression status

Change 137864 on 2003/12/16 by rramsey@rramsey_xenos_linux_orl

Change emulator so param cache reads for params not exported by the VS
still show up in sq_sx_pcaddr.
Fix cf_resource_change logic in the cfs so it catches the clause boundary
where a cf instr with only tex instr gets sent to the alu cfs.

Change 137773 on 2003/12/16 by vromaker@vromaker_r400_linux_marlboro

- fixed a predicate override bug: pred_overide is driven from the done bits if
the previous operation was a waterfall, but it must only be driven for the first
instruction following a waterfall. The bug occurred on back-to-back waterfalls
where the pred_override was being driven for all cycles of the second waterfall.
- this fix caused r400sq_gpr_index_01 to pass

Change 137753 on 2003/12/16 by danh@danh_r400_win

Status update.

Change 137701 on 2003/12/15 by rramsey@rramsey_xenos_linux_orl

Add new _sf (single-file) versions of PLI routines that allow trackers
to only open their dump files one time.
Modify a few trackers and models to use the new _sf routines to verify
they are working.
Fix a problem with the cfsm not ignoring clause boundaries for unexecuted
predicate control flow instr.

Change 137566 on 2003/12/15 by donaldl@donaldl_xenos_linux_orl

Added tracker for RSP to SX data.

Change 137560 on 2003/12/15 by danh@danh_r400_win

Updated status.

Change 137415 on 2003/12/15 by jcarroll@jcarroll_r400_win

Picked tests

Change 137401 on 2003/12/15 by rramsey@RRAMSEY_P4_r400_win

update with weekend's results

Change 137238 on 2003/12/12 by vromaker@vromaker_r400_linux_marlboro

took a few more tests

Change 137205 on 2003/12/12 by mearl@mearl_r400_win

updated status

Change 137188 on 2003/12/12 by rramsey@RRAMSEY_P4_r400_win

update with latest regression results

Change 137166 on 2003/12/12 by vromaker@vromaker_r400_linux_marlboro

- increased the depth of the sq-vc request fifo; this is a temporary fix while the
mini and mega dec signals from the VC are added to the vc_rp_sp dump file

Change 137165 on 2003/12/12 by mearl@mearl_r400_win

updated status

Change 137146 on 2003/12/12 by vromaker@vromaker_r400_linux_marlboro

updated status

Change 137105 on 2003/12/12 by mmantor@mmantor_xenos_linux_orl

<fixed bug in the emu for redundancy control, added new input to the sq called sx_sp_alloc_table_free >

Change 137104 on 2003/12/12 by mmantor@mmantor_xenos_linux_orl

<This changed changed SQ and SX top level ports by added thread_type from sq_alloc through the sx so tracker at sx_rb works correct and fixed some other minor bugs>

Change 136917 on 2003/12/11 by mearl@mearl_r400_win

updated status

Change 136893 on 2003/12/11 by mearl@mearl_r400_win

updated status

Change 136888 on 2003/12/11 by bhankins@bhankins_xenos_linux_orl

Change defined "C1" switch to "NEW_BC" in sx rtl and related vcpp files

Change 136871 on 2003/12/11 by mearl@mearl_r400_win

updated status

Change 136867 on 2003/12/11 by rramsey@rramsey_xenos_linux_orl

don't reset current_context at eo_rt load

Change 136773 on 2003/12/10 by danh@danh_r400_win

Updated status.

Change 136758 on 2003/12/10 by bhankins@bhankins_xenos_linux_orl

fix ifdef/endif mismatch

Change 136713 on 2003/12/10 by vromaker@vromaker_r400_linux_marlboro

updated status

Change 136691 on 2003/12/10 by bhankins@bhankins_xenos_linux_orl

1. Add ability for both r400 and xenos versions of sx to coexist
2. Rewrite memory read mux select logic in sx_bc_if.v for better synthesis
3. Add quad_x and quad_y signals to BC interface.
4. Update 'copy_virage_' files to reflect memory updates
5. Change 'ENABLE_SX_TO_BC' compile switch to 'C1'
6. Remove obsolete code (sx_export_buffers_common.v logic is now in sx_rb_if.v)
7. Update virage .cnt files

Change 136672 on 2003/12/10 by mearl@mearl_r400_win

updated status

Change 136596 on 2003/12/09 by vromaker@vromaker_r400_linux_marlboro

- added a couple wire names for ppb read data in cfs
- added fsdb dump for tbtrk_sq_vtx_rs_input in tb_sqsp
- changed checking of predicate to registered version in above trk to fix false mismatch
- bit 95 of vc/tp instruction was wired to 0 causing a mismatch, so
  I changed it to the actual instruction bit 95 (which is only used by the sq)

Change 136574 on 2003/12/09 by danh@danh_r400_win

Updated r400sc_rts_* status

Change 136557 on 2003/12/09 by mmantor@mmantor_xenos_linux_orl

<fixed allocation counter for ea and cleaned up controls for rest of the counters and fixed a bug in the spi_sp tracker by removing delay on sq_sp_simd_id because of pipelining the vertex and pixel input data>

Change 136424 on 2003/12/09 by mearl@mearl_r400_win

update status

Change 136358 on 2003/12/09 by vromaker@vromaker_r400_linux_marlboro

updated status for r400sq_auto_wrapping_memories_01 (test issue)

Change 136334 on 2003/12/09 by rramsey@RRAMSEY_P4_r400_win

update dot2add status, take more tests

Change 136332 on 2003/12/09 by mmantor@FL_mmantorLT_r400_win

<took test with >2 exports>

Change 136326 on 2003/12/09 by mearl@mearl_r400_win

took a few tests

Change 136192 on 2003/12/08 by mearl@mearl_r400_win

Removed more SC pipe disable tests.

Change 136174 on 2003/12/08 by danh@danh_r400_win

Updated r400sc_* status

Change 136166 on 2003/12/08 by llefebvr@llefebvre_laptop_r400_emu

working on r400sp_coissue_add_01.cpp

Change 136153 on 2003/12/08 by vromaker@vromaker_r400_linux_marlboro

added my name by a few tests

Change 136141 on 2003/12/08 by mearl@mearl_r400_win

Removed pipe disable tests, renamed and moved to the ROM block

Change 136135 on 2003/12/08 by rramsey@rramsey_xenos_linux_orl

Add a bit to pix thread counter to handle larger thread buffer.

Change 136102 on 2003/12/08 by rramsey@RRAMSEY_P4_r400_win

update 'sorted by type' page with latest results

Change 136063 on 2003/12/08 by mmantor@mmantor_xenos_linux_orl

<another synthesis issue>

Change 135995 on 2003/12/08 by danh@danh_r400_win

Updated r400sc_sp_sample_cntl* status

Change 135987 on 2003/12/08 by rramsey@RRAMSEY_P4_r400_win

update with status from 12/8/2003

Change 135983 on 2003/12/08 by dclifton@dclifton_r400

Updated for new sq rams

Change 135975 on 2003/12/08 by mmantor@mmantor_xenos_linux_orl

<fixed leda errors for synthesis>

Change 135943 on 2003/12/07 by vromaker@vromaker_r400_linux_marlboro

- connected resource management register to thread buffers
  (programmable thread buffer size)
- fixed typo and leda error in sq_vtx_ctl

Change 135932 on 2003/12/07 by rramsey@rramsey_xenos_linux_orl

fix a problem with vizq_start events and how they cause state locks in the tb.
this should fix the vgt_event tests

Change 135879 on 2003/12/05 by mmantor@mmantor_xenos_linux_orl

<fixed a synthesis problem during elaboration in the sq_input_arb.v and fixed a problem with redunancy so that both vertex and pixel input controllers would send simd_id with there respective request to the spi. This change renamed a top level port between the sq and sp sq_sp_interp_simd_id changed to sq_sp_simd_id >

Change 135600 on 2003/12/05 by bhankins@bhankins_xenos_linux_orl

1. add behavioral support for sx to bc interface. Disabled.
2. fixed bug in alloc/dealloc block to hold off resetting alloc bit until the last bank of memory is read for a particular address.

3. fixed bug in alloc/dealloc block where free logic was searching all 256 locations of the buffer when only 128 are enabled.
4. connect SX_SQ_free_export_address_buf to indicate last quad of memory export has been created and written to the detailed quad fifo.
5. fix minor bug in sx-rb interface logic that would have shown up with larger export buffer.

Change 135598 on 2003/12/05 by smoss@smoss_crayola_linux_orl_regress

removed reference to internal tracker

Change 135584 on 2003/12/05 by rramsey@rramsey_xenos_linux_orl

absolute address mode (const_addr_mode = 3'b001) should apply to all src constants

Change 135537 on 2003/12/04 by vromaker@vromaker_r400_linux_marlboro

- increased size of thread buffers: vtx from 16 to 32 threads, pix from 48 to 64 thread
- fixed gpr dealloc bug that resulted in reduced performance
- testbench and tracker changes were made to support the larger number of threads
- emualtor change (separate checkin) was also made for the bigger thread buffers

Change 135234 on 2003/12/04 by mmantor@mmantor_xenos_linux_orl

<1. wired simd2 and simd3 pipe_disable_vtx for proper vertex steering with 2 and 3 simds.
2. Improved usaged of gpr input (Interp/Vtx data) port by pipline the vtx input controller
data and changing input arbiter. Still need to make a changed in sq_pix_ctl to remove
busy signal on last 4clockcycle so interleaving can be tighter and in both machines load
to the thread buffer sooner once commmitted.
3. changed SIMD2_PRESENT_TMP to SIMD2_PRESENT in sq code, still needs to be removed from
tb_sqsp and check other parts of the design such as the sp and rsp.>

Change 135151 on 2003/12/03 by mearl@mearl_xenos_linux_orl

Took out the SC_SP_last_quad signal from the top of the SC
and SPI.

Change 135088 on 2003/12/03 by rramsey@rramsey_xenos_linux_orl

Add a unique vc/tp update for pending bits to the thread status regs
so they don't clobber each other.
Don't compare fetch constants for VC mini_fetches.
Change tex_instr_seq so fetch type (vc/tp) is determined based on the fetch opcode
rather than thread type.

Fix a problem in sx_export_control when a pos free_done happened on the same clk as a
pos_dec when exporting aux vectors.

Change 134818 on 2003/12/02 by llefebvr@llefebvr_r400_linux_marlboro

Now using the sq_aiq_bX_rts signal to drive the alu active counter. It used to be driven by ais_busy which was also high when doing TP and VC fetches.

Change 134653 on 2003/12/02 by bhankins@bhankins_xenos_linux_orl

- replace rf implementation of the quad buffer fifo from the sc with an implememtation using hs ram
- add new star processor to support hs ram

Change 134460 on 2003/12/01 by smoss@smoss_crayola_linux_orl_regress

removed bad path for sx_defines.v

Change 134408 on 2003/12/01 by rramsey@rramsey_xenos_linux_orl

Fix HI/LO instruction split

Change 134404 on 2003/12/01 by bhankins@bhankins_xenos_linux_orl

1. Added SX_INDEX_SIZE and SX_INDEX_SIZE_EQ_8 defines to the rtl, defined in sx_defines.v and set equal t
2. Moved sx_defines.v to parts_lib/src/common
3. Renamed sx inputs that connect to the STAR processors to match the names in the TST module

Change 134359 on 2003/11/30 by mmantor@mmantor_xenos_linux_orl

<removed delay in/outs from tb_sqsp>

Change 134302 on 2003/11/28 by mmantor@mmantor_xenos_linux_orl

<remove delay chain from sq>

Change 134126 on 2003/11/26 by donaldl@donaldl_xenos_linux_orl

Updated tbmod_fake_rb to create multiple file pointers based on thread_id[5:0] and
thread_type. Needed because the quad_index[7:0] can come in out of order from the
SX. The quad_index[7:0] and op bit are stored in a fifo and eventually sent back
to the SX.

Change 134100 on 2003/11/26 by donaldl@donaldl_xenos_linux_orl

Removed pred_kill_type and pred_kill_valid input signals since no longer used.
(ie. they are being delayed internally.

Change 133815 on 2003/11/25 by bhankins@bhankins_xenos_linux_orl

Increase depth of color buffer to 256. Only the first 128 locations are enabled for now.

Change 133588 on 2003/11/24 by dclifton@dclifton_r400

Removed delay chain I/O from SP's

Change 133300 on 2003/11/21 by vromaker@vromaker_r400_linux_marlboro

timing fix - stopped using any unregistered status read bits

Change 133275 on 2003/11/21 by mearl@mearl_xenos_linux_orl

1. Took out delay chain in the SC and SC_B blocks.
chip_sc.tree
chip_sc_b.tree
sc.v
sc_b.v
tb_sqsp_sc_iter.v
2. Timing related changes.
sc_packer.v
sc_packer_pkg.v
3. Real-Time tracker changes
sc_block_model.cpp
sc_interp.cpp
sc_types.h
out_compare.v
tb_sc.v
tbtrk_sc.v

Change 133264 on 2003/11/21 by bhankins@bhankins_xenos_linux_orl

1. Restructured sx to have an sx-rb interface block sx_rb_if, readying it for
a similar sx_bc_if block for xenos.
2. Removed delay chain
3. Changed input quad fifo to dum_mem for now.
4. Removed some unused signals.
5. Changed pav.tree per Vivian's request to change test signal name.

Change 132894 on 2003/11/19 by rramsey@rramsey_xenos_linux_orl

Fix SQ_VC dec signals in tb_sqsp.
Change tbtrk_sqvc so it does not compare fetch addr for mini fetches.
Fix problem in tex_instr_seq that was allowing mini fetches to start
out of phase.
Add more info to msgs from pcdata tracker to tell which set of pc data
is mismatching. Also turn off sx1 compare since it is redundant now that

all the sx data comes from usx_0.

Change 132675 on 2003/11/18 by danh@danh_xenos_linux_orl

Added the tbtrk_sq_sx_pcaddr tracker.

Change 132667 on 2003/11/18 by danh@danh_xenos_linux_orl

Initial Release.

Change 132649 on 2003/11/18 by vromaker@vromaker_r400_linux_marlboro

- alu_instr_seq timing fixes for constant store read: first the register stage
on the offset was moved after the sum2 adder; then the init_done_bits signal
was changed from a combinational ACS state machine output to a registered
one-bit state machine output to help the path to the new sum2 register
- thread buff status read timing fix - moved the status read back one cycle by
sending the unregistered, rotated request vector to the arbiter and registering
the winner out of the arbiter; the output of the status read mux was
then registered

Change 132516 on 2003/11/18 by rramsey@rramsey_xenos_linux_orl

Add a mova test to the sq regression.
Change no_inc in pix_ctl to use sr version instead of nxt value
out of the ppb.
Fix instr base calc in rbbm_if so rt/nrt determination is correct.
Stop vec_grp tracker from comparing pix auto_count cycles.

Change 132219 on 2003/11/16 by smoss@smoss_crayola_linux_orl_regress

<Orlando Hardware Regression Results >

Change 132123 on 2003/11/14 by rramsey@rramsey_xenos_linux_orl

Fix a bug in aluconst_mem related to rt constant reads.
Fix const_map_cntl so deallocate_cnt gets updated correctly when alloc and
context_done happen on same clk.
tb_sqsp was missing some primitive boundaries in the pkr for RT prims.

Change 131826 on 2003/11/13 by rramsey@rramsey_xenos_linux_orl

get rid of ifndefs so vcs will compile

Change 131814 on 2003/11/13 by dclifton@dclifton_xenos_linux_orl

Added register for undriven signal

Change 131722 on 2003/11/13 by rramsey@rramsey_xenos_linux_orl

Add capability to dump Cadence shm instead of fsdb. Enabled by defining
DUMP_SHM in tb_sqsp/vcsopts.f file

Change 131537 on 2003/11/12 by llefebvr@llefebvr_r400_linux_marlboro

1) added register stage to line up pred_override bits with SP phase
2) made the waterfall/predicated override an or instead of an and.

Change 131465 on 2003/11/11 by donaldl@donaldl_xenos_linux_orl

When in the VS_EVENT state and going to IDLE, update d_sp_sel[3:0] as a default
based on disable_vtx_3,2,1,0 instead of 0. This is to fix a bug where the correct
o_sp_vsr_valid bit was not being set because the disable simd flags were not being
considered (when going from VS_EVENT to IDLE).

Change 131241 on 2003/11/11 by kmeekins@kmeekins_xenos_linux_orl

Removed event window from VC counters.

Change 131082 on 2003/11/10 by kmeekins@kmeekins_xenos_linux_orl

tb_vc.v

Fixed instantiation of vc now that delay is removed.

sq_fetch_arb.v

Changed the bus width of vc_mini_count_q to accomidate the +2 modification.

vcmi_requestor.v

Increased the uvcmi_input_fifo FIFO depth to 8.
Added the FIFO full to the performance monitor.

tp.blk,
vc.v,
vc_perf_config.txt,
vc_perfmon.v,
vcmi.v

Added the FIFO full for the vcmi_input_fifo to the performance monitor.

Change 130763 on 2003/11/07 by llefebvr@llefebvr_r400_linux_marlboro

Reverting timing fix that broke r400sq_const_index_04.cpp test.

Change 130661 on 2003/11/07 by rramsey@rramsey_xenos_linux_orl

Another attempt to keep the pc_out_ppb from overflowing

Change 130571 on 2003/11/06 by llefebvr@llefebvr_r400_linux_marlboro

This fixes the bad pix/vtx GPR input arbitration performance counter.

Change 130421 on 2003/11/06 by bhankins@bhankins_xenos_linux_orl

- sq-sx thread id added to sq output and into and through the sx
  - updated sx-rb trackers to use sq-sx thread id
  - removed obsolete code from sx
  - fixed sx bug where an ea from one export to memory was resetting the valid bits for the other export to memory

Change 130346 on 2003/11/05 by danh@danh_xenos_linux_orl

Removed spi delay_in and delay_out ports.

Change 130127 on 2003/11/04 by vromaker@vromaker_r400_linux_marlboro

- instruction writes to the different SIMD memories now happen independently and no longer wait for all SIMD memories to be available

Change 130094 on 2003/11/04 by rramsey@rramsey_xenos_linux_orl

Fix scalar tracker so it compares all 128 bits based on write masks
It was only comparing the lower 32 bits based on bit 0 of the write mask

Change 130079 on 2003/11/04 by rramsey@rramsey_xenos_linux_orl

Couple of timing fixes for aiq and cfs
Fix a bug in the rbbm if that was allowing map copies to happen before memory writes
Fix a problem in the testbench that was causing some incompletes

Change 130072 on 2003/11/04 by rramsey@rramsey_xenos_linux_orl

Update tracker to work with new sp_sx dump file that has all free-done entries as unique lines between exports

Change 129980 on 2003/11/03 by smoss@smoss_crayola_linux_orl_regress

some housekeeping and removed bad path

Change 129723 on 2003/11/01 by vromaker@vromaker_r400_linux_marlboro

- fixed pix ctl output buffer overwrite bug
- backed timing fix out of status reg and pix thread buff

Change 129444 on 2003/10/30 by llefebvr@llefebvr_r400_linux_marlboro

Fixing dangling wires in the sq related to performance module.
Fixing shader due to Kill opcode assembler change.
Fixing trakcer problem in the TB_SQSP when autocount vtx is on.

Change 129408 on 2003/10/30 by rramsey@rramsey_xenos_linux_orl

Move some continuous assignments into always blocks to help sim time
Rework cfs_rtr/arb_xfc path to help timing
Fix a problem with detecting serialize for the cf state machine

Change 129348 on 2003/10/30 by mearl@mearl_xenos_linux_orl

Added two primitive interpolation back in.

Change 129259 on 2003/10/29 by danh@danh_xenos_linux_orl

- spi_interp_ctl IJ buffer changed from one 16x200 memory to two 16x100 memories.
  - added additional SQ_SP_interp_qd[0:1]_prim_sela signals to improve spi input timing.

Change 129213 on 2003/10/29 by llefebvr@llefebvr_r400_linux_marlboro

Added VC_PERF_ACTUAL_STARVED performance counter in the SQ.

Change 129150 on 2003/10/29 by llefebvr@llefebvr_r400_linux_marlboro

Increasing VC mini count to l1_fifo_size +2.

Change 129066 on 2003/10/28 by vromaker@vromaker_r400_linux_marlboro

- added vtx input optimization for autocount on and continued off
- fixed initialization problem for vtx autocount
- made pix thread buff timing fixes: reduced load on status read data bit 19, which is the event bit, and also tried to reduce the load on pop_thread (part of the same path) in the status register
- backed out a timing fix in alu_instr_seq that was causing a mova test to fail
- fixed the AUTO_COUNT_SIZE definition

Change 128816 on 2003/10/27 by llefebvr@llefebvr_r400_linux_marlboro

Adding VC performance counters in the SQ.
Removed the SX->RB warnings on non-initialized GPR channels.

Change 128675 on 2003/10/27 by smoss@smoss_xenos_linux_orl

combined ncverilog and vcs simulators to one build

Change 128659 on 2003/10/27 by donaldl@donaldl_xenos_linux_orl

Delayed rom_rsp_shift*_* mux shift selects 1 clk to fix synthesis timing.

Change 128656 on 2003/10/27 by donaldl@donaldl_xenos_linux_orl

Changed vc_req's and tex_req's dependencies on vc_pending_q and tp_pending_q.

Change 128647 on 2003/10/27 by rramsey@rramsey_xenos_linux_orl

Change ais so PS src sel gets priority over PV
Add predicated jumps and calls to cfs
Fix fetch_type connection in sq and tex_instr_seq

Change 128645 on 2003/10/27 by llefebvr@llefebvr_r400_linux_marlboro

Incrementing the number of in flight testure requests from 6 to 7.

Change 128601 on 2003/10/27 by mmantor@mmantor_xenos_linux_orl

<Enable SQ use of 128 locations in export memmory instead of 112 locations. Also added counters in sq arbiter to give priority to instruction pipe that has the fewest instructions when both control flow machines are available. This changlist reguires both an emulator and hardware rtl code updates>

Change 128592 on 2003/10/26 by danh@danh_xenos_linux_orl

Changed sc_rt_valid to fix the condition when end_of_prim and end_of_vector do not occur at the same time, the sc_packer will send real time fill quads.

Change 128526 on 2003/10/24 by mearl@mearl_xenos_linux_orl

Took out two prim per clock to get regression to pass.

Change 128393 on 2003/10/24 by llefebvr@llefebvr_r400_linux_marlboro

This should fix the instruction count being off. The bad machine (cfs) was used to determine the thread type and hence some pixel shader instructions were counted as vertex ones and vice versa.

Change 128365 on 2003/10/24 by mearl@mearl_xenos_linux_orl

Added 2 primitive interpolation in SQ and SPI. Fixed a bug in sx_parameter_cache. Fixed synthesis
bugs in SC.

Change 128209 on 2003/10/23 by vromaker@vromaker_r400_linux_marlboro

- timing fixes for constant store read address

Change 128195 on 2003/10/23 by rramsey@rramsey_xenos_linux_orl

Fix a problem with yield_optimize

Change 128048 on 2003/10/23 by llefebvr@llefebvr_r400_linux_marlboro

Fixed problem in the active counters when both pixels and vertexes were processing at the same time.

Change 128019 on 2003/10/23 by rramsey@rramsey_xenos_linux_orl

go back to prev version

Change 127895 on 2003/10/22 by vromaker@vromaker_r400_linux_marlboro

- timing fixes for gpr alloc

Change 127872 on 2003/10/22 by rramsey@rramsey_xenos_linux_orl

fixes for MT3 functions

Change 127861 on 2003/10/22 by llefebvr@llefebvr_r400_linux_marlboro

Fixing TP and VC sync stalls for both pixel and vertex threads.

Change 127742 on 2003/10/22 by llefebvr@llefebvr_r400_linux_marlboro

Removed the warnings from the sp->sx trackers and sx->sp.
Now emulator is always executing the scalar instruction even in the case of a 3 operand vector opcode. This is to match with random shaders.

Change 127730 on 2003/10/22 by rramsey@rramsey_xenos_linux_orl

Fix a bug with start_of_clause

Change 127580 on 2003/10/21 by danh@danh_xenos_linux_orl

Changed any_pred_hi and any_pred_lo generation, now the predicate and valid bits are now related to the thread that the CFS is working on.

Change 127397 on 2003/10/20 by llefebvr@llefebvr_r400_linux_marlboro

Added an event window for pixels. There was a problem in the global event window as if both pixels and vertexes were turned on at the same time, as soon as one went off it was turning off the whole window. This fixes pixel counters being 0 for some tests.

Change 127325 on 2003/10/20 by vromaker@vromaker_r400_linux_marlboro

- updated VC injector to handle multi-cycle returns (the number of cycles, 1 to 4, is
  read from the vc_rp_sp.dmp file)

Change 127313 on 2003/10/20 by dclifton@dclifton_r400

Updated to testbench changes.

Change 127269 on 2003/10/19 by rramsey@rramsey_xenos_linux_orl

Change behave mem_model in spi so its read dly matches the real mem
Send interp_valid and ij_line 1clk early to account for 2clk read dly
Fix spi_sp tracker so it works with early valid
Change thread_buf and cfs machines so only fetches can modify the
fetch pending bit. The alu machines only read the value out of the buffer.
Get rid of a bunch of extra 'else' clauses

Change 127091 on 2003/10/17 by rramsey@RRAMSEY_P4_r400_win

udpate spreadsheet with 10/17/03 results
modify script so it automatically handles reports with/without runtime

Change 127079 on 2003/10/17 by smoss@smoss_xenos_linux_orl

initialized memory controller for sc and sx to allow real memories to work in tb_sqsp

Change 126983 on 2003/10/16 by vromaker@vromaker_r400_linux_marlboro

fixed code that was causing a latch in synthesis

Change 126908 on 2003/10/16 by rramsey@rramsey_xenos_linux_orl

absolute modifier for constants should apply to all source constants

Change 126823 on 2003/10/15 by rramsey@rramsey_xenos_linux_orl

Add sqvc tracker to gc testbench when running with orlando trackers
Rework some of the alu/tex constant logic to get rid of the bug that
was allowing threads to start processing before all of the constants for
their context had been loaded.

Change 126796 on 2003/10/15 by vromaker@vromaker_r400_linux_marlboro

- hooked up the new alu_arb_policy and tx_cache_sel register bits (but
  temporarily tied the tx_cache_sel input to the vtx thread buff low
  since it is being incorrectly set to 1 by Primlib)

Change 126483 on 2003/10/13 by mearl@mearl_xenos_linux_orl

Fix One Prim Per Clock bug in sq_ptr_buff. Revert changes in sq_pix_ctl to make
2 prim interp changes easier. Put known primdata data on all quads across packer
to iterator interface. Fix dumps for no_inc_pix_cnt signal.

Change 126450 on 2003/10/13 by donaldl@donaldl_xenos_linux_orl

Delayed SQ_SX_sp_simd_id an extra clock to line up for redudancy use.

Change 126362 on 2003/10/13 by rramsey@rramsey_xenos_linux_orl

Change sq_sp_interp dump so it contains all of the pass_count and wrap passes
through the interpolator
Add spi_sp tracker (enabled with ENABLE_SPI_TRACKER define)

Change 126324 on 2003/10/13 by dougd@dougd_r400_linux_marlboro

Added logic to generate read enables for the 4 map rams in sq_aluconst_rams.v
Added SQ_CONTEXT_MISC_YEILD_OPTIMIZE register to sq_rbbm_interface.v

Change 126234 on 2003/10/10 by vromaker@vromaker_r400_linux_marlboro

- added export arbiter module that will limit the number of color buffer export
  threads to one every 4 clocks
- hooked up the export blocker outputs and commented out the previous export
  blocking code
- added export alloc arbiter inputs to exp_alloc_ctl module so that the buf_avail
  counter will be updated by the export allocs
- added logic to support the export arbiter to the vertex and pixel thread buffers
- added logic to support the export arbiter to the thread arbiter
- separated the export alloc request out of the alu request logic in the status register,
  and added an output for the export alloc request

Change 126226 on 2003/10/10 by cbrennan@cbrennan_r400_emu

Release from my emu branch: texture stacks for TP as well.
Leda rule tweaks
add more .rg files

Change 125806 on 2003/10/09 by cbrennan@cbrennan_r400_release

Temporarily reduce the num SQ_TP vectors in flight back to 6 until fifo overflows can be fixed.

Change 125780 on 2003/10/09 by bhankins@bhankins_xenos_linux_orl

update sx test inputs to match the established convention

Change 125697 on 2003/10/08 by dougd@dougd_r400_linux_marlboro

fixed bug in eqn for *sync_alu_stall

Change 125660 on 2003/10/08 by rramsey@rramsey_xenos_linux_orl

Fix compile warnings for sq (several missing ports)
Fix compile warning in sx_parameter_caches
Fix SQ_SP_fetch_simd_sel so it lines up with the data coming out of the GPRs

Change 125598 on 2003/10/08 by dougd@dougd_r400_linux_marlboro

Expanded the read back mux for rbbm diagnostic reads
to include the extra memories for SIMD2 and SIMD3.

Change 125550 on 2003/10/08 by rramsey@rramsey_xenos_linux_orl

Increase sq_tp_maxcount from 6 to 7
Fix a problem with the simd mux for vtx_alloc_size in export_alloc
Fix a problem with pc_alloc_free_cnt in export_alloc (alloc and dealloc on same clk
was broken)
Make alu ctl_flow and instr trackers work with multiple simd's
Also change these trackers to use common code for pix/vtx by selecting the type with
a parameter

Change 125540 on 2003/10/08 by dclifton@dclifton_r400

Added needed include files. Strange how these compiled before this.

Change 125509 on 2003/10/07 by dougd@dougd_r400_linux_marlboro

change perfcounters alu(0/1)_fifo_empty_simd* to count
alu(0/1)_stall_simd* instead.

Change 125370 on 2003/10/07 by mearl@mearl_xenos_linux_orl

Fixed the SQ bug when bad pipe exists before a good pipe. Also, updated
the RT trackers in the SC testbench.

Change 125278 on 2003/10/07 by dougd@dougd_r400_linux_marlboro

Added a new state register, vc_fifo_depths_l1_req_fifo_depth to
sq_rbbm_interface.v and wired it up to the compare logic for
vc_mini_count_q in sq_fetch_arb.v.

Corrected a typo in sq_vtx_ctl.v that affected synthesis.

Change 125260 on 2003/10/07 by dclifton@dclifton_r400

Updates for a couple of fifos in sq and new block in sp

Change 125059 on 2003/10/06 by rramsey@rramsey_xenos_linux_orl

Fix sq_sx file read in tb_sqsp
Add new tracker for shader writes to gpr
Add myself to failing regression email list

Change 124864 on 2003/10/03 by rramsey@rramsey_xenos_linux_orl

add some missing wire declarations

Change 124850 on 2003/10/03 by rramsey@rramsey_xenos_linux_orl

move an adder in front of a register and change to a fifo with registered
outputs to help timing

Change 124792 on 2003/10/03 by dougd@dougd_r400_linux_marlboro

Removed all references to SIMD1_DISABLE in sq.v and sq_rbbm_interface.v.

Added 32 new performance counters: many are for SIMD2 and SIMD3 but
other existing counters were expanded to differentiate between vertex
and pixel counts. There are now 95 performance counters in the sq.

Change 124774 on 2003/10/03 by smoss@smoss_crayola_linux_orl_regress

re-enabled behavioral memories until real memories are working

Change 124741 on 2003/10/03 by bhankins@bhankins_xenos_linux_orl

fix name on sx test pin

Change 124738 on 2003/10/03 by smoss@smoss_crayola_linux_orl_regress

<Orlando Hardware Regression Results >

Change 124634 on 2003/10/02 by rramsey@rramsey_xenos_linux_orl

adding cond_pred optimize to control flow seq

Change 124434 on 2003/10/01 by mmang@mmang_xenos_linux_orl

1. Turned on 3 simds in emulator (sc_interp.cpp,
   sq_block_model.cpp, and user_block_model.cpp).
2. Turned on 3 simds in rtl (sc_packer.v,
   tb_sqsp.v, and vgt.v).
3. Fixed bug in chip_vc.tree to get SQ_VC_simd_id
   and TC_VC_simd hooked up correctly.

4. Fixed bug in sc_packer.v related to having a 2
bit simd_id_sel.

Change 124292 on 2003/10/01 by rramsey@rramsey_xenos_linux_orl

Change sq_vgt_rtr to be driven based on fifo full, rather than by the vsr
load state machine

Change 124203 on 2003/10/01 by dougd@dougd_r400_linux_marlboro

The four existing SYNC_STALL counters were separated into
(8) pix and vtx stall counters.
The two ALU INSTRUCTION ISSUED counters were made to increment
by 1,2,3 or 4.
The two CF INSTRUCTION ISSUED counters were made to increment
by 1,2,3,4,5 or 6.

Added `ifdef's to sq_perfmon_wrapper for SIMD1, SIMD2, SIMD3.

perfmon event window:
An enable for the performance counters is generated by events received
from the VGT and/or SC which create a window of time when the counters
will be active. All of the perf counters are now controlled by this enable.

Change 123984 on 2003/09/30 by bhankins@bhankins_xenos_linux_orl

change names of sx i/o ROM_MCn_disable signals

Change 123966 on 2003/09/30 by smoss@smoss_xenos_linux_orl

using real memories for sqsp

Change 123952 on 2003/09/30 by mmantor@mmantor_xenos_linux_orl

<added changes for 2 prim interpolation to the spi and sq and all top level interconnects, and sq_sx_sp_simd_id for redundancy control, and all changes to test bench as well as some ncverilog error messages. Some other misc top level clean up>

Change 123918 on 2003/09/29 by rramsey@rramsey_xenos_linux_orl

Change tp_sqsp dump to use FMT_32_32_32_32_FLOAT
Remove a monitor from tbtrk_sc for now since it is broken for ONEPPC
Need to register the if inputs to aiq since they are put in the fifo
one clk after the transfer
Fix the exec_sm so it is 4 clks even when switching clauses
Remove one clk of latency on tp_dec from fetch_arb
Fix the strap bits in sq.v so the tp and vc cfs and if machines get
two read cycles out of 8 when we have two instruction stores
Change the tp_sq dec input and force the tp_sp format in tb_sqsp

Fix the tif so its state machine is 4 clks between clauses and change
it so 0 count execs can be merged into the instruction ahead of them
Fix the tex_instr_seq for the case where tp_dec happens on the same
clk the fcs state machine kicks off (instr were getting dropped)
Check in Scott's vgt change to clamp vtx_reuse based on good pipes

Change 123798 on 2003/09/29 by donaldl@donaldl_xenos_linux_orl

Temporary hook-up of SQ_SX_interp_2prim to zero going to SX until
SQ changes for 2 prims is complete.

Change 123755 on 2003/09/29 by mearl@mearl_xenos_linux_orl

Fix for timing problems, submitting new memories, using real memories for regressions.

Change 123528 on 2003/09/26 by llefebvr@llefebvr_r400_linux_marlboro

The sp->sx, sq->tp and sq->vc trackers now all use the post steered valid bits to know what is valid. Thus they are now compatible with the redundant pipe. They should track correctly in any bad pipe configuration. They however don't compare the RSP data for now (waits for the HW implementation)

Change 123515 on 2003/09/26 by bhankins@bhankins_xenos_linux_orl

- add sx_redundancy.v to hierarchy to try and improve on timing
  - add EXP_BUF_112_DEEP switch. comment out in sx_defines.v to enable
    all 128 locations of the color export buffer to be used
  - add ONE_STAR_PROCESSOR switch. comment out in sx_defines.v to use
    two star processors.
  - add support for thread id and thread type for debug.
  - misc changes for timing which don't change the logic.

Change 123485 on 2003/09/26 by dougd@dougd_r400_linux_marlboro

I removed these files prematurely.

Change 123462 on 2003/09/26 by dclifton@dclifton_r400

disabled USE_BEHAVE_MEM. Changed 8x104 ram in sq to 8x105.

Change 123343 on 2003/09/25 by dougd@dougd_r400_linux_marlboro

adding the x105 virage memories and deleting the x104 used in the sq_vc_skid_buf

Change 123331 on 2003/09/25 by dougd@dougd_r400_linux_marlboro

usq_alu01_state_mem is used twice as the instance name so I changed
the 2nd one to usq_alu23_state_mem.

Change 123260 on 2003/09/25 by mmang@mmang_xenos_linux_orl

1. For Vivian E., added new simd memories and star patch in/out wires.
2. In vertex thread buffer, fixed bug in simd3 alu state registers.
3. In pixel thread buffer, fixed bug in simd2/3 cf state read data.
4. Adjusted simd id bus width for sq to tp tracker.
5. In sq.v, added vertex shader and pixel shader constant base and
   size connections to simd2/3 alu instruction sequencers.

Change 123113 on 2003/09/24 by llefebvr@llefebvr_r400_linux_marlboro

Fixed the autocount pixel timing by removing 5 pipeline registers in the SQ control path. Also fixed the counter's with back to 17 bits (from 19) int both the vertex and pixel path such that when it hits the SP it is of the correct 23 bits width (17 bits count + 2 bits phase + 4 bits index). This fixes r400vgt_multi_pass_pix_shader_01 at the sqspsx testbench level.

Change 123082 on 2003/09/24 by mearl@mearl_crayola_linux_orl

tb files updated for ONE_PRIM_PER_CLOCK, bug fix in interpolators for ONE_PRIM_PER_CLOCK

Change 123076 on 2003/09/24 by donaldl@donaldl_xenos_linux_orl

Connected ROM block redundancy signals.
Added sq export address buffer support.

Change 122865 on 2003/09/23 by dougd@dougd_r400_linux_marlboro

fixed typo

Change 122699 on 2003/09/23 by dougd@dougd_r400_linux_marlboro

fix typo (change blocking to non-blocking assignment)

Change 122683 on 2003/09/23 by mearl@mearl_crayola_linux_orl

One primitieve per clock changes in the back of the SC and front of the SQ. Right now, the ONE_PRIM_PER_CLOCK define in
header.v and SC_SQ_interface.v are needed for this change. Will update this to ONEPPC, since this already exists in
header.v. Also, the sim.cfg file does not have an ifdef, so is hardcoded to one prim per clock.

Change 122558 on 2003/09/22 by dougd@dougd_r400_linux_marlboro

1. changed sq_stdrfsdks2p8x104cm1sw0 to sq_stdrfsdks2p8x105cm1sw0 in sq_vc_skid_buf.v
2. added timing fixes to sq_aluconst_mem.v, sq_aluconst_rams.v and sq_instruction_store.v

Change 122520 on 2003/09/22 by vromaker@vromaker_r400_linux_marlboro

timing fixes - added registers for vs and ps base and size after the
context register read mux

Change 122402 on 2003/09/20 by mmang@mmang_crayola_linux_orl

1. Added simd2 and simd3 to code.
2. Added simd2 to synthesized code.
3. In sq.blk and sq_rbbm_interface, added
   DB_READ_MEMORY, DB_WEN_MEMORY_2, and DB_WEN_MEMORY_3
   to SQ_MISC_DEBUG register.
4. In header.v, turned on SIMD2_PRESENT.
5. In sc_packer.v, turned on SIMD2 but don't use it
   with SIMD2_PRESENT_TEMP.
6. In sq_aluconst_mem.v, sq_aluconst_top.v, sq_cfc.v,
   and sq_instruction_store.v, hooked up DB_WEN_MEMORY_2
   and DB_WEN_MEMORY_3 to appropriate SIMD2/3 memories.
7. In sq_export_alloc.v, handle position/main export id
   and parameter cache thread base for simd2/3. Be able
   to handle one type down simd0/1 and a different type
   down simd2/3 on the same clock.
8. In sq_pix_ctl.v and sq_vtx_ctl.v, multiple simd
   gpr_alloc blocks return different acks, gpr bases,
   and gpr maxes.
9. In sq_exp_alloc_ctrl.v, handle position/main export
   buffer management. Be able handle one type down
   simd0/1 and a different type down simd2/3 on the same
   clock.
10. In sq_pix_thread_buff.v and sq_vtx_pix_thread_buff.v,
   added muxing and memories to handle status bits, cfs
   state, and alu state. Simd2 mirrors simd0, while
   simd3 mirrors simd1.
11. In sq_status_reg.v, added simd2/3 arb requests and
   status bit writing from simd2/3.
12. In tb_sqsp.v, fixed some bugs related to pspv_wr_en,
   pred_override, const_addr, and const_valid hook ups.
13. In tbtrk_spsx.v, SIMD_PRESENT conditional delaying
   and management of thread_id and thread_type for
   tracker.
14. In tbtrk_sq_pix_rs_input.v and tbtrk_sq_vtx_rs_input.v,
   temporary klug to hook up b0b1_predicate instead of
   predicate.
15. In tbtrk_sq_sp_vec_gpr.v, added simd2/3 tracking of
   gpr_int_wen interface.
16. In sq_tex_instr_queue.v, get gpr_max from appropriate
   simd data.<enter description here>

```
Change 121731 on 2003/09/17 by rramsey@RRAMSEY_P4_r400_win

        add runtime to report
        update spreadsheet with 9/17/2003 results

Change 121629 on 2003/09/16 by danh@danh_crayola1_linux_orl

        Removed XY pipe delay, XY data is now processed by the interpolators

Change 121559 on 2003/09/16 by tien@tien_r500_emu

        Reverse order of TP (vfetch and tfetch) const

Change 121537 on 2003/09/16 by smoss@smoss_crayola_linux_orl_regress

          increasing interface idle timeout for randoms

Change 121348 on 2003/09/15 by dougd@dougd_r400_linux_marlboro

        1. corrected the trigger events for VTX_SWAP_IN, VTX_SWAP_OUT,
          PIX_SWAP_IN, PIX_SWAP_OUT, CONSTANTS_USED_SIMD0 and
CONSTANTS_USED_SIMD0.
        2. made event counters for these used multibit increment values
        3. added  "+incdir+$PARTS_LIB/src/gfx/sp" to vcs_top.ini to pick up
          sp_defines.v included in sq_ais_output.v

Change 121332 on 2003/09/15 by rramsey@rramsey_crayola_linux_orl

        Change pix_ctl so deallocs with real pixel vectors don't free param
        cache space until interpolation is almost complete
        Wire up the vc_sp valid signals correctly
        Fix sx_sp_pcdata tracker

Change 121292 on 2003/09/15 by vromaker@vromaker_r400_linux_marlboro

        fixed incorrect loading of loop indices from the thread buffer into
        the ctl flow sequencer; this was causing a problem with the test
        r400sq_const_index_07

Change 121278 on 2003/09/15 by dclifton@dclifton_r400

        Added to SQ include directory list

Change 121219 on 2003/09/14 by smoss@smoss_crayola_linux_orl_regress

        <Orlando Hardware Regression Results >

Change 121157 on 2003/09/13 by smoss@smoss_crayola_linux_orl_regress
```

```
          xenos updates

Change 121065 on 2003/09/12 by donaldl@donaldl_crayola_linux_orl

        Registered ROM_EN_RSP and ROM_PIPE_SEL[3:0].

Change 120910 on 2003/09/12 by donaldl@donaldl_crayola_linux_orl

        Removed SPtoSQ kill_type and kill_valid signals and added them internally
        in the SQ.  Done to save some gates and also to avoid having to add
        redundancy logic to them.

Change 120887 on 2003/09/12 by bhankins@bhankins_crayola_linux_orl

        -  Add sx_mem_export.v module to capture pixel addresses and
             calculate rb id values for use in export to memory.
          -  Add support for redundancy logic.  Inputs are currently
             tied low in tb_sqsp.v and chip_sx.tree.
          -  Add non-synthesizable logic to route thread id and thread
             type from sq through sx and out to rb for test.  Allows
             tracker to identify export to memories, and to distinguish
             between them.  Tied low in chip_sx.tree and tb_sqsp.v
             All associated I/O and logic is qualified on `ifdef SIM.
          -  Remove the register in sx_export_control_common.v that was
             requiring some signals on the sq alloc interface to be present
             one clock before the valid.  Now, all sq_sx_exp_ signals are
             expected to be valid only when sq_sx_exp_valid == 1.
          -  Add a register in the generation of the final pixel address
             value for export to memory, to try and improve on timing.

Change 120645 on 2003/09/11 by rramsey@rramsey_crayola_linux_orl

        Remove some unused defines
        Add reset condition for primdata pipe stages in qdpr_proc
        Fix a bug with tp_count in fetch_arb when running with the VC
        Increase loop_cnt for vc inject in tb_sqsp

Change 120592 on 2003/09/10 by vromaker@vromaker_r400_linux_marlboro

        changed SQ_hs_bclk, TST_SQ_rf_star_wrck, TST_SQ_hs_star_wrck so they
              are defined without the [0:0] range

Change 120510 on 2003/09/10 by vromaker@vromaker_r400_linux_marlboro

        fix for SQ_VC_simd_id typo

Change 120426 on 2003/09/10 by donaldl@donaldl_crayola_linux_orl

        Added redundancy logic.
```

```
Change 120423 on 2003/09/10 by donaldl@donaldl_crayola_linux_orl

        Added redundancy logic.

Change 120397 on 2003/09/10 by rramsey@rramsey_crayola_linux_orl

        Add code to keep the vc and tp inject routines from clobbering each other
        Fix vc inject routine so it handles formats that require double returns

Change 120296 on 2003/09/09 by dougd@dougd_r400_linux_marlboro

        added `include "register_addr.v"

Change 120270 on 2003/09/09 by llefebvr@llefebvr_r400_linux_marlboro

        Now reading the SIMD_ID from the dump in the tracker. Not doing anything with it
however. It is just read in order to get to the valid data after it.

Change 120190 on 2003/09/09 by dougd@dougd_r400_linux_marlboro

        changed SQ_RB_event to SQ_RB_event_pulse and declared as output from sq.v

Change 120087 on 2003/09/08 by dougd@dougd_r400_linux_marlboro

        Fixed 2 bugs in Real Time address logic in aluconst.
        Added correct default value for INST_BASE_VTX in sq_rbbm_interface.v
        Fixed bug in Real Time write data buffer in sq_instruction_store.v
        Added missing input/output declarations for SIMD2 & SIMD3 signals to
sq_aluconst_top.v
        Clean up missing SIMD2, SIMD3 wire declarations in sq.v for the aluconst, is and cfc

Change 119982 on 2003/09/08 by vromaker@vromaker_r400_linux_marlboro

        added defaults to case statements

Change 119853 on 2003/09/06 by rramsey@rramsey_crayola_linux_orl

        Changes to make quad processing resources programmable

Change 119747 on 2003/09/05 by danh@danh_crayola1_linux_orl

        Removed SQ_SP_interp_mode, SQ_SP_interp_buff_swap, added all SPI Redundant SP
ports/connections.

Change 119736 on 2003/09/05 by danh@danh_crayola1_linux_orl

        removed SQ_SP_interp_mode, SQ_SP_interp_buff_swap, added SQ_SP_interp_simd_id
for Redundant SP
```

```
Change 119733 on 2003/09/05 by danh@danh_crayola1_linux_orl

        removed SQ_SP_interp_mode, added SQ_SP_interp_simd_id for Redundant SP
capability.

Change 119457 on 2003/09/04 by dclifton@dclifton_r400

        added sq_export_blocker to makefile
        Fixed TP_SP_data_valid signal

Change 119422 on 2003/09/04 by mmang@mmang_crayola_linux_orl

        removed vc_sp for now

Change 119294 on 2003/09/03 by vromaker@vromaker_r400_linux_marlboro

        - instatiation of sq export blocker at sq top level
        - thread buffer timing fix related to status read/export count update

Change 119195 on 2003/09/03 by vromaker@vromaker_r400_linux_marlboro

        new file for arbitrating between exporting threads

Change 119127 on 2003/09/02 by dougd@dougd_r400_linux_marlboro

        Added the extra memories and their support to the instruction and
        constant stores to support 4 SIMD's. These memories and their
        required wiring and control are instantiated with `ifdef and use
        the SIMDn_PRESENT macros defined in header.v
        Removed the use of SIMD1 macro.

Change 118878 on 2003/08/30 by rramsey@rramsey_crayola_linux_orl

        fix a deadlock condition between the input arb and vtx input controller

Change 118743 on 2003/08/29 by viviana@viviana_crayola2_syn

        Configuration file to build the virage memories with a register in the
          320x32 cfc memory.

Change 118694 on 2003/08/29 by rramsey@rramsey_crayola_linux_orl

        changes for random backpressure

Change 118622 on 2003/08/28 by llefebvr@llefebvr_r400_emu_montreal
```

Modified the Orlando trackers to only compare valid channels. This replaces the 0xDEADDEAD values we had previously. Note that any uninitialized channel will generate a tracker warning still.
Modified interfaces are:

1) SX->SP parameter cache data
2) SP->SX
3) SX->RB

I left alone the SX->PA interface as we did not have problems over it. The qualifiers are there however if anyone wants to do it.

Change 118589 on 2003/08/28 by vromaker@vromaker_r400_linux_marlboro

- fix for loop index clamping and constant address generation (both index and offset relative)
- changed the connection of the real time bit such that it now goes directly from the AIQ to the
  AIS output mux (and not thru the AIS)
- sq_tests.simple_reg_indexing tests now pass

Change 118581 on 2003/08/28 by dclifton@dclifton_r400

tied the upper bit of sq_tp_trk_simd_id low.

Change 118490 on 2003/08/28 by dclifton@dclifton_r400

Clean up of unused signals, fix of STAR signals in sp.v

Change 118397 on 2003/08/27 by smoss@smoss_crayola_linux_orl_regress

<Orlando Hardware Regression Results >

Change 118215 on 2003/08/26 by vromaker@vromaker_r400_linux_marlboro

changed define for SQ_VC_MINI_MAXCOUNT from 16 to 32

Change 118200 on 2003/08/26 by rramsey@rramsey_crayola_linux_orl

Increase number of clks the tp_sq inject routine can loop through input data
Fix a problem with the sx_rb color tracker when the sx sends 0 mask quads, or
the rb kills quads

Change 118130 on 2003/08/26 by dclifton@dclifton_r400

Added tbtrk_sqvc, fixed vector engine assignments.

Change 118128 on 2003/08/26 by dclifton@dclifton_r400

Added definable # of simd's to sp.

Change 117957 on 2003/08/25 by dougd@dougd_r400_linux_marlboro

Fixed some wiring errors in the wrapper that prevented some counters from working.

Change 117706 on 2003/08/22 by mmantor@mmantor_crayola_linux_orl

<added new ports and/or expanded to two bits to vgt, sq, and pa for simd_id with modifications to their test benches and added
ifdefs with bad pipe signals to input of vgt, replaced SIMD1 macro with SIMD1_PRESENT macro in the SC files>

Change 117704 on 2003/08/22 by mmantor@mmantor_crayola_linux_orl

<Fixed conflict between vec_3op_no_swap and scalar_const_op to control swizzle correctly for the scalar engine and deliever the special gpr read address created in the sq_ais_output block>

Change 117631 on 2003/08/21 by vromaker@vromaker_r400_linux_marlboro

- fix for VC_SQ_data_rdy (this was being asserted too often, but did not
  cause any of the tests to fail...)

Change 117627 on 2003/08/21 by vromaker@vromaker_r400_linux_marlboro

VC tracker added to tb_sqsp

Change 117504 on 2003/08/21 by mmang@mmang_crayola_linux_orl

1. Increased simd_id wires to 2 bits throughout SQ. SQ external
   interfaces are still only 1 bit.
2. Made SQ simd 1 blocks conditional based on SIMD1_PRESENT in
   header.v. Realigned some code in anticipation of SIMD2 and SIMD3.

Change 117311 on 2003/08/20 by rramsey@rramsey_crayola_linux_orl

Changes to sc for 4 qd/clk picker in KILL_ALL_PIXELS mode
Check in sc memory updates for Vivian
Add some missing connections in sqsp to fix compile warnings
Go to a global define for all trackers to control x vs 0 mismatch/warning (MISMATCH_X_VS_0)

Change 116887 on 2003/08/18 by dougd@dougd_r400_linux_marlboro

restore the `ifdef USE_BEHAVE_MEM that was removed for
testing of virage behavioral models.

Change 116795 on 2003/08/15 by vromaker@vromaker_r400_linux_marlboro

adding sq-vc tracker (not debugged yet - just checking in working copy)

Change 116380 on 2003/08/13 by mmang@mmang_crayola_linux_orl

1. Added separate gpr allocation/deallocation
   management for multiple simds (sq_gpr_alloc,
   sq_exit_sm, sq_pix_thread_buff, sq_status_reg,
   sq_vtx_thread_buff, sq_pix_ctl, and sq_vtx_ctl)
2. Made thread_arb poll cfs rtr on a 4 clock
   interval in order to ensure the arbiters
   stayed in phase between simds.
3. Created new interface signal between
   thread_arb and export_alloc to lock export_id
   and parameter cache base for each simd. In
   addition, created registers for these values
   for each simd in order to ensure they got
   allocated in order.
4. In ais_output, used simd to mask pix_ctl gpr
   writes to different simds.
5. In tb_sqsp, added simd_id and gpr write address
   to texture latency fifo to help trackers and
   read inject return files.
6. In tex_instr_queue, grab appropriate gpr_max
   based on simd id.

Change 116303 on 2003/08/13 by danh@danh_r400_win

Updated failing tests status.

Change 115781 on 2003/08/11 by rramsey@RRAMSEY_P4_r400_win

update sq status
add runtime column to report so it works with the spreadsheet script

Change 115728 on 2003/08/10 by rramsey@rramsey_crayola_linux_orl

Change SQ to hold off popping the RBBM skid fifo while map copies are in
progress. This fixes the problem where gfx_copy writes were being missed
if they were less than 8 clks apart.
Get rid of extra write into RBBM skid fifo for reads, and instead zero out
we and re out of fifo if it's empty. The fifo was overflowing if the filling
entry was a read, since one additional entry was getting pushed.
sx_sp_pcdata tracker now ignores 4f5eaddf (unwritten pc locations)
Fix a problem in the sqsp testbench that was causing rbbm writes to be dropped
if the sq exerted back pressure.

Change 115620 on 2003/08/08 by dougd@dougd_r400_linux_marlboro

1. change all hs virage memories & files to have subword size in name
2. added diagnostic write enable from rbbm interface register to the modules
   with extra memories to support multiple SIMDs

Change 115595 on 2003/08/08 by dougd@dougd_r400_linux_marlboro

fixed the path for the real time bit down the alu pipeline
to reach the constant and instruction stores.

Change 115581 on 2003/08/08 by rramsey@RRAMSEY_P4_r400_win

update sq status

Change 115492 on 2003/08/07 by mmang@mmang_crayola_linux_orl

change order of include paths for register_addr.v

Change 115430 on 2003/08/07 by danh@danh_r400_win

Updated status (lines 271-284).

Change 115426 on 2003/08/07 by dclifton@dclifton_r400

Added another block

Change 115274 on 2003/08/06 by smoss@smoss_crayola_win

monitor strange ncsim errors

Change 115254 on 2003/08/06 by smoss@smoss_crayola_win

fixing deaddead

Change 115241 on 2003/08/06 by dougd@dougd_r400_linux_marlboro

1. corrected the connections to sq_perfmon_wrapper to enable the
ALU active counters.

2. changed a few 1 bit vector declarations ([0:0]) to scalar
on SQ outputs because it caused errors in synthesis.

Change 115159 on 2003/08/06 by rramsey@rramsey_crayola_linux_orl

Change sq_alu_instr_seq so gpr_rd_en is not asserted when reading constants
Changes to thread_arb, ctl_flow_seq, and status_reg to get mem exports flowing

Change 115122 on 2003/08/06 by rramsey@RRAMSEY_P4_r400_win

Update with Aug6 sanity results and add a new worksheet that has failures sorted by failure type

Change 115115 on 2003/08/06 by smoss@smoss_crayola_linux_orl_regress

Randy's keeping me honest

Change 115114 on 2003/08/06 by rramsey@rramsey_crayola_linux_orl

add some missing dummy dump files

Change 115049 on 2003/08/05 by rramsey@RRAMSEY_P4_r400_win

Put some comments on all of the failing tests so we can try to bin the issues for debugging

Change 115047 on 2003/08/05 by rramsey@rramsey_crayola_linux_orl

Add register to hold pipe disable bits to tb_sqsp
Hook sx instance up to correct set of RBBM signals in tb_sqsp
Increase depth of sc state avail fifo since some events need
to go through that path
Change sx pa tracker to always opens its files so it doesn't
cause hangs when the files are empty
Add deaddead and a selectable x_vs_0 mismatch disable (reports
a warning rather than a mismatch) to tbtrk_sx_rb.v

Change 114774 on 2003/08/04 by rramsey@RRAMSEY_P4_r400_win

update sqspsx status

Change 114706 on 2003/08/04 by danh@danh_r400_win

Updated r400sq_* status.

Change 114427 on 2003/08/01 by smoss@smoss_crayola_linux_orl_regress

added rb_sx dump

Change 114404 on 2003/08/01 by amys@amys_r400_regress_linux

changes made to fix running ncsim using Orlando trackers

Change 114305 on 2003/07/31 by vromaker@vromaker_r400_linux_marlboro

cleaned up the path of ism_state down through the
instruction pipelines and removed the defparams used in the
multiple instantiations of several modules.

Change 114167 on 2003/07/31 by danh@danh_r400_win

Updated the r400sq_* status.

Change 114159 on 2003/07/31 by rramsey@RRAMSEY_P4_r400_win

update status. remove some CP tests that don't anything at sqsp.

Change 113990 on 2003/07/30 by rramsey@rramsey_crayola_linux_orl

Changes to support real time prims.
Tests that draw rt only now drive sc inputs
RBBM stream is held off while each rt prim processes so
rt code/const/params are not clobbered

Change 113953 on 2003/07/30 by danh@danh_r400_win

Updated r400sq_* status.

Change 113550 on 2003/07/28 by dougd@dougd_r400_linux_marlboro

added define+virage_ignore_read_addx to support virage behavoral models

Change 113548 on 2003/07/28 by dougd@dougd_r400_linux_marlboro

Added missing register stage in memory address path that caused
memory failures only with the virage behavoral model.

Change 113503 on 2003/07/28 by rramsey@RRAMSEY_P4_r400_win

update sq stats

Change 113302 on 2003/07/25 by danh@danh_r400_win

Updated r400sq_* status.

Change 113293 on 2003/07/25 by rramsey@RRAMSEY_P4_r400_win

update sq status

Change 113286 on 2003/07/25 by vromaker@vromaker_r400_linux_marlboro

- a few more fixes for SQ_VC/TP interfaces; the sq mini-regress now passes
  with the VC turned on

Change 113223 on 2003/07/25 by rramsey@rramsey_crayola_linux_orl

uncomment driver for SQ_SP_interp_xyline

Change 113207 on 2003/07/25 by danh@danh_r400_win

Updated the r400sq_* status.

Change 113039 on 2003/07/24 by danh@danh_crayola1_linux_orl

Changed src_c_const_addr_rel generation so it matches the emulator code.

Change 112899 on 2003/07/24 by danh@danh_crayola1_linux_orl

Changed src_c_const_addr_rel generation.

Change 112882 on 2003/07/24 by rramsey@RRAMSEY_P4_r400_win

update sqspsx status

Change 112600 on 2003/07/23 by rramsey@rramsey_crayola_linux_orl

Change sx-rb trackers so they always open their files at time 0,
that way they don't cause hangs for tests that don't hit any quads
Hook up the real pixel mask in the sx_rb color tracker

Change 112375 on 2003/07/22 by vromaker@vromaker_r400_linux_marlboro

- fixed VC interface counter

Change 112335 on 2003/07/22 by danh@danh_r400_win

Updated the r400sq* status.

Change 112289 on 2003/07/22 by dclifton@dclifton_r400

Updated staging registers in sp_macc.
Revised sp_scalar_lut.
Test signals connected.

Change 112108 on 2003/07/21 by rramsey@RRAMSEY_P4_r400_win

update with 07/21 status and some comments on the failing tests

Change 112073 on 2003/07/21 by vromaker@vromaker_r400_linux_marlboro

- fix for SQ_VC interface
- TP_SQ_dec was hooked up to the interface counter
- timing fix in vtx thread buffer
- simd_num connected thru ptr buff and pix ctl to pix thread buff
- performance fix in pix ctl

Change 112034 on 2003/07/19 by rramsey@rramsey_crayola_linux_orl

Change vcs build script so cover is off by default
Get rid of some compile warnings in tb_sqsp
Change sx_rb color tracker so it doesn't use the sx_rb_quad dump
to get pixel masks

Change 111986 on 2003/07/18 by dougd@dougd_r400_linux_marlboro

Added dummy mems for all virage memorie that didn't already have them.
Moved memory data output register in sq_cfc.v into the memory and dummy memory.
Replaced all virage memories, etc. to get the memory needed for sq_cfc.v

Change 111905 on 2003/07/18 by ygiang@ygiang_r400_pv2_marlboro

added: new perf counters for sq hardware

Change 111807 on 2003/07/18 by mmantor@mmantor_crayola_linux_orl

<added new dummy file for test cases that needed it>

Change 111736 on 2003/07/17 by mmang@mmang_crayola_linux_orl

Added sp->sx export arbitration between multiple simd engines.
Added register after instr_start OR of multiple simd engines by
taking unregistered signal out of sq_ais_output.

Change 111732 on 2003/07/17 by rramsey@RRAMSEY_P4_r400_win

Update with regression results, plus a couple of my own

Change 111726 on 2003/07/17 by smoss@smoss_crayola_linux_orl_regress

modified $value$plusargs to keep cadence happy

Change 111692 on 2003/07/17 by danh@danh_r400_win

Updated the r400sq* status.

Change 111650 on 2003/07/17 by rramsey@rramsey_crayola_linux_orl

Add pasx done to testbench timeout logic

Change 111628 on 2003/07/17 by smoss@smoss_crayola_linux_orl_regress

changed tbmod_fake_pa for ncsim because all requests weren't occurring this was also true for vcs but sim was passing. changed buildt for nc to not run a sim after a compile

Change 111612 on 2003/07/17 by moev@moev2_r400_linux_marlboro

Clean up files no longer used by the verification flow

Change 111603 on 2003/07/17 by moev@moev2_r400_linux_marlboro

SQ changes to test Virage's HS memories.

Change 111419 on 2003/07/16 by rramsey@rramsey_crayola_linux_orl

Connect TST_awt_enable to vc_skid_buf and wire it up to the top level

Change 111381 on 2003/07/16 by rramsey@rramsey_crayola_linux_orl

Fix compile result check in buildtb
Tie off sx related done signals when the sx is not there
and spit them out if it is there and the tb hangs
Don't source sx_sp_pcdata stimulus when using live sx
Remove extra ifdef

Change 111353 on 2003/07/16 by bhankins@bhankins_crayola_linux_orl

when the sx is present, include the sx trackers in on the decision to stop the simulation

Change 111345 on 2003/07/16 by rramsey@RRAMSEY_P4_r400_win

Fix the update script to handle 'run time' being reported
Redo the last status update to the spreadsheet since 'run time'
caused all the fields to get shifted

Change 111342 on 2003/07/16 by smoss@smoss_crayola_linux_orl_regress

<Orlando Hardware Regression Results >

Change 111317 on 2003/07/15 by mmang@mmang_crayola_linux_orl

Blocking/non-blocking fix found by synthesis.

Change 111305 on 2003/07/15 by smoss@smoss_crayola_win

update

Change 111303 on 2003/07/15 by rramsey@rramsey_crayola_linux_orl

allow pa/sx requests before the rbbm file is empty

Change 111280 on 2003/07/15 by rramsey@rramsey_crayola_linux_orl

need to wait for vc_done if serialize and vc_pending

Change 111275 on 2003/07/15 by rramsey@rramsey_crayola_linux_orl

add SX_BLOCK_SIM so the sx trackers know where they are running

Change 111132 on 2003/07/15 by smoss@smoss_crayola_linux_orl

just copying randy

Change 111123 on 2003/07/15 by rramsey@rramsey_crayola_linux_orl

had a typo in the vc_pending logic

Change 111107 on 2003/07/15 by smoss@smoss_crayola_linux_orl

updated

Change 111093 on 2003/07/15 by smoss@smoss_crayola_linux_orl

decapitating tb_sc

Change 111008 on 2003/07/14 by dougd@dougd_r400_linux_marlboro

added logic to support programmable memory size for texconst and
aluconst stores.

Change 110899 on 2003/07/14 by rramsey@rramsey_crayola_linux_orl

change tp/vc pending bits so they look at tgt_instr_str_vc_q bits to
determine what type of fetch is being issued

Change 110886 on 2003/07/14 by rramsey@rramsey_crayola_linux_orl

mask off serial bit for first instruction of a clause.
this change fixes e2blit_src_8888 and probably some other hanging
e2/cp tests

Change 110884 on 2003/07/14 by rramsey@RRAMSEY_P4_r400_win

update with latest regression results

Change 110880 on 2003/07/14 by rramsey@rramsey_crayola_linux_orl

Add back in a signal declaration to fix the no SX build
Move some signals to the other half of a REMOVE_SX ifdef

Change 110669 on 2003/07/12 by smoss@smoss_crayola_linux_orl_regress

removed errant else

Change 110640 on 2003/07/12 by mmantor@mmantor_crayola_linux_orl

<1. Enlarge export memories for performance fill rate (emulator, sq, sx, rb, ferret gc, tb_sqsp, tb_sx)
2. Fix Sx diff engine (interpolators) for shift bug with added guard bit
3. Fix compile/src code problem with s-blocks memories
4. Added the sx to tb_sqsp by default, can still disable by macro
5. Added mode to tb_sqsp and tb_sx to run interfaces at max rate
6. Initialized state in vc to allow cp surface synchronizer micro code to invalidate tc/vc
7. Added test signals to sc.v, sc_b.v, sq, sp, spi, sx and testbenches
THIS CHANGES REQUIRES THE RELEASE OF SC, SC_B, SQ, SPI, SP, SX, RB, src/chip/chip_**.tree files,
parts_lib/sim/test/gc/vcs_top.ini, gc/tb_sqsp/tb_sx updates and the emulator togeather
>

Change 110512 on 2003/07/11 by mmang@mmang_crayola_linux_orl

Fix for Vivian for synthesis in loop i07 and i15.

Change 110467 on 2003/07/11 by llefebvr@llefebvr_r400_emu_montreal

Disabling the COND_EXEC_PRED optimization. a COND_EXEC_PRED in the SQ is now threated like a regular EXEC. We can re-enable this optimization in the future by putting the thread back to the RS BEFORE making the predicate compare because now we are comapring a dirty predicate bit set and it causes corruptions. This fixes mova_test.cpp TEST_CASE=pMova_const.

Change 110451 on 2003/07/11 by dclifton@dclifton_r400

Fixed typo for spi ram compile

Change 110401 on 2003/07/11 by viviana@viviana_crayola2_syn

Changed the sq/vc 103 memory to 104.

Change 110310 on 2003/07/10 by viviana@viviana_crayola2_syn

Changed the vc memory to 104 bits wide, deleted the 103 memory and rebuilt all
memories with latest version of virage.

Change 110177 on 2003/07/10 by rramsey@rramsey_crayola_linux_orl

Changes to get simd_id piped down the vertex side and into the thread
buffer. Also only write the active simd's gprs and mux pipe_disable bits.
The memory in sq_vc_skid_buf increased by 1 bit, so this will require
a new memory to be checked in before running without USE_BEHAVE_MEM.

Change 110083 on 2003/07/09 by dougd@dougd_r400_linux_marlboro

added data output mux to select between the two memories (SIMD1, SIMD0)

for RBBM diagnostic reads. The mux is controlled by a rbbm register bit
in the SQ_DEBUG_MISC register.

Change 110066 on 2003/07/09 by vromaker@vromaker_r400_linux_marlboro

- fixed a bug in tex instr seq related to back-to-back constant reads

Change 110035 on 2003/07/09 by moev@moev2_r400_linux_marlboro

Changed the HS Star Processor connections to match the clients. In particular BiraFail & Err_pip_or

Change 109951 on 2003/07/09 by llefebvr@llefebvr_r400_emu_montreal

Fixing yet another mova problem when the mova is not back to back with it's use and there is only one waterfall pass, PVPS detection wasn't re-enabled correctly. Fixes mova_tests.cpp TEST_CASE=mova512_nop_check

Change 109814 on 2003/07/08 by vromaker@vromaker_r400_linux_marlboro

- contains RT bit connection from pix input ctl to pix thread buff
- added SQ_TP_simd_id output to top level

Change 109777 on 2003/07/08 by vromaker@vromaker_r400_linux_marlboro

Change 109679 on 2003/07/08 by llefebvr@llefebvr_r400_emu_montreal

Fixed r400sp_mova_tests.cpp TEST_CASE=mova512.

The PVPS detection was rightly disabled during the waterfall but wasn't re-enabled for the following instructions of the clause. I used the waterfall_done signal to re-enable the PVPS detection after the waterfalling.

Change 109671 on 2003/07/08 by vromaker@vromaker_r400_linux_marlboro

- updated tex instr seq to sync to the texconst phase
- changed fetch arb to output both the mega grant and the mini
grant to the tex instr seq

Change 109590 on 2003/07/07 by viviana@viviana_crayola2_syn

Corrected another non-blocking assignment to blocking in a combinational logic block.

Change 109565 on 2003/07/07 by viviana@viviana_crayola2_syn

Corrected non-blocking assignments to blocking in combinational block.

Change 109466 on 2003/07/07 by dougd@dougd_r400_linux_marlboro

fixed error in bit width of ais_real_time

Change 109126 on 2003/07/03 by dougd@dougd_r400_linux_marlboro

pipelined the Real Time bit from the pix thread buffer down through
both arbiters, the vc, tex and alu instruction pipelines to the alu,
tex and cfc constant stores to enable reading the real time constants.

Change 109043 on 2003/07/03 by vromaker@vromaker_r400_linux_marlboro

made all loop counter variables unique for sythesis

Change 108947 on 2003/07/02 by dclifton@dclifton_r400

Updated makefile for latest changes. Fixed testbench test signals into SP and SPI.

Change 108763 on 2003/07/01 by llefebvr@llefebvr_r400_emu_montreal

Updates for r400sq_const_index_0x.cpp

Change 108760 on 2003/07/01 by llefebvr@llefebvr_r400_linux_marlboro

Fixed r400sq_const_index_03.cpp. Now works on the SQSP testbench. Still has issues on the GC because of bad ferret/cp ring buffer synchronization.

Fixed:

1) Bad clamping of the address register in the SP
2) Bad error handling of an out of range address in the SQ.

Change 108744 on 2003/07/01 by vromaker@vromaker_r400_linux_marlboro

- registered winner_ack out of thread arb for timing
- connected correct instruction store read output based on SIMD1
  for VC ctl flow instruction reads; now SQ_VC interface appears to
  be driven correctly
- minor change to tb_sqsp (commented out random stall for TP_SQ_fetch
  stall, which no longer exists)

Change 108676 on 2003/07/01 by dougd@dougd_r400_linux_marlboro

generated trigger signals for SIMD0,SIMD1 perfmon counters

Change 108585 on 2003/06/30 by rramsey@rramsey_crayola_linux_orl

hook up the sx_rb_quad_mask signals to the fake_rb's
not sure how this was working at all with the live SX

Change 108536 on 2003/06/30 by smoss@smoss_crayola_linux_orl_regress

removed rand function warning

Change 108524 on 2003/06/30 by dougd@dougd_r400_linux_marlboro

generate read enable for sq_hs_sms_sq_shsd1_320x96cm4 in sq_texconst_mem
and read enable for sq_stdrfsdks2p64x32cm4sw0 in sq_texconst_rams

Change 108511 on 2003/06/30 by rramsey@rramsey_crayola_linux_orl

changes for new sp top level

Change 108315 on 2003/06/27 by mmang@mmang_crayola_linux_orl

Qualify constant address register write using constant waterfalling mask

Change 108250 on 2003/06/27 by rramsey@rramsey_crayola_linux_orl

left some signals out of a sensitivity list

Change 108222 on 2003/06/27 by smoss@smoss_crayola_linux_orl_regress

I have too many i's

Change 108208 on 2003/06/26 by dclifton@dclifton_r400

Changes to get the tb_sqsp to work in modelsim

Change 108188 on 2003/06/26 by mmang@mmang_crayola_linux_orl

For pixel quads, enable all pixels of a quad when any pixel is hit
for gpr write enables and constant address waterfalling sequencing.
Another update will fix constant address register writing.

Change 108140 on 2003/06/26 by rramsey@rramsey_crayola_linux_orl

Split src_swizzle out of SQ_SP_instr bus so fetch swizzle can be
driven during unused phase
Add interp_xyline from SQ to SPI to drive read address for xy buffer
Clean up some compile warnings in sc_iter
Change the existing macc to handle the swizzle being driven for all
4 phases and add the fetch address swizzling
Fix param_gen and gen_index pipeline length around the interpolators
Replace src_c_swizzle.z with src_c_swizzle.x for all instructions
other then MULADD and CNDx
Fix the generation of init_cycle_cnt_q in sq_pix_ctl for interpolation
involving param_gen and gen_index params
Add compares for SQ_SX_export_mask_we and SQ_SX_kill_mask to tbtrk_spsx

Fix the fetch_addr swizzle generation for vertex fetches (need to use
[31:30] instead of [27:26])
Fix a bug in sq_vtx_ctl related to gpr allocation (size requested was
off by a clock)

Change 108063 on 2003/06/26 by viviana@viviana_crayola2_syn

Regenerated the high speed memories to add two instances of the 1280x128 and two instances of the
4096x96.

Change 108024 on 2003/06/26 by mmantor@FL_mmantorLT_r400_win

remove template file having problems in ncverilog

Change 107822 on 2003/06/25 by rramsey@RRAMSEY_P4_r400_win

and another syntax error

Change 107820 on 2003/06/25 by rramsey@RRAMSEY_P4_r400_win

fix decimal vs hex problem

Change 107817 on 2003/06/25 by fhsien@fhsien_r400_LT

correct syntax error

Change 107801 on 2003/06/25 by grayc@grayc_crayola2_linux_orl

fix syntax

Change 107757 on 2003/06/25 by mmantor@mmantor_crayola_linux_orl

< 1. sq_alu_instr_seq.v - Use the Queue pop signal to qualify last_in_clause
and last_in_shader out of the queue.
2. sq_target_instr_fetch.v - Fixed a buf in the the target_instruct_fetch
write to the queue to prevent dropping last_in_shader and last_in_clause
if the queue is full when first trying to send instruction. >

Change 107717 on 2003/06/24 by mmantor@mmantor_crayola_linux_orl

<added new regression test for cyl_wrap and changed vcs for texconst mem and fixed wrap bug in controller during interpolation and added a dum mem config for the texconst memory >

Change 107579 on 2003/06/24 by dougd@dougd_r400_linux_marlboro

ncverilog will error with
output [0:0] SQ_SP_instruct_start

wire SQ_SP_instruct_start
because it considers the 1st declaration a vector and
the 2nd one a scalar.

Change 107389 on 2003/06/22 by mmang@mmang_crayola_linux_orl

1. made change sp_vector.v to grab pred/kill results
   a clock sooner since Vic a register delay to
   sp_scalar_lut.bvrl. May have to change back later.
2. Took away register delay in sq_ais_output to account
   for extra register needed for muxing and registering
   both simd engines for SQ_SX_sp signals.
3. In sq_alu_instr_seq.v, backed out Laurent's previous
   fix for constant waterfalling and made different change
   where ism registers are loaded based on ais_start
   instead of ais_rtr. With waterfalling, the ais_rtr
   does not happen early enough for ism registers to be
   available for AIS state machine.
4. In sq_export_alloc.v, added connections for second simd
   engine to handle sx export allocation and deallocation.
5. In sq.v, added muxing between simd0 and simd1
   sq_ais_output for SQ_SX signals.
6. In sq_exp_alloc_ctrl.v, added simd1 connections for
   sx export control logic.
7. In sq_pix_thread_buff.v and sq_vtx_thread_buff.v, added
   A) Simd1 logic for ALU memory write (register delayed
      simd1 information to avoid overlap with simd0)
   B) Appropriate read mux for simd0/simd1 for control
      flow memory (based on status simd num).
   C) Added simd1 status register write data connections.
8. In sq_status_reg.v, added connections and muxing for second
   simd engine status bits write.
9. Added a variety of connections for simd1 to tb_sqsp.v.
10. Added delay pipe for thread_id and thread_type for simd1
   in order to correctly track sp to sx interface. (tbtrk_spsx.v)
11. Fixed bug in sx related to using correct export id during
   free done process of pixel to rb buffers
   (sx_export_control_common.v)

Change 107266 on 2003/06/20 by vromaker@vromaker_r400_linux_marlboro

reverted a change that was made for VC testing (and that did not work correctly)

Change 107174 on 2003/06/20 by vromaker@vromaker_r400_linux_marlboro

- swapped PS and ID gpr write phases

Change 107015 on 2003/06/19 by viviana@viviana_crayola2_syn

Re-ran cover on the high speed memories to add fuse_box318 files previously deleted.
Also deleted fuse_box29 files no longer used.

Change 107009 on 2003/06/19 by smoss@smoss_crayola_linux_orl_regress

update

Change 106949 on 2003/06/19 by smoss@smoss_crayola_linux_orl_regress

removed sq_tp_stall signal in anticipation of new sq_tp interface

Change 106751 on 2003/06/18 by danh@danh_r400_win

Updated r400sq_* status.

Change 106611 on 2003/06/17 by danh@danh_crayola1_linux_orl

Changed the cfs_return_addrs_q[51:0] generation so the correct cfs_return_addr[3:0]_q order
will be written into the thread buffer CFS mem when a thread is returned to the thread buffer.

Change 106597 on 2003/06/17 by rramsey@RRAMSEY_P4_r400_win

more status

Change 106528 on 2003/06/17 by rramsey@rramsey_crayola_linux_orl

hook up iterator_SP_cntx0 so realtime works correctly

Change 106375 on 2003/06/16 by danh@danh_r400_win

Updated the r400sq* status.

Change 106357 on 2003/06/16 by rramsey@rramsey_crayola_linux_orl

fix latency of tp/sp signals in tb_sqsp after tp_formatter change
clean up the fetch swizzle warning msg in tb_sqsp
add new memory to sq/tb.f
fix fech_swizzle signal width in tex_instr_seq

Change 106293 on 2003/06/16 by vromaker@vromaker_r400_linux_marlboro

code fix to prevent latches

Change 106277 on 2003/06/16 by viviana@viviana_crayola2_syn

Extra bit added to pixel state data.

Change 106273 on 2003/06/16 by danh@danh_crayola1_linux_orl

Changed TB_TP_REQ_FIFO_DEPTH (128 to 256) &
TB_TP_REQ_FIFO_ADDR_WIDTH (7 to 8) to resolve fifo overflow.

Change 106191 on 2003/06/14 by viviana@viviana_crayola2_syn

48x154 memory changed to 48x155.

Change 106190 on 2003/06/14 by viviana@viviana_crayola2_syn

Changed the width of the state memory to 155 bits.

Change 106078 on 2003/06/13 by rramsey@RRAMSEY_P4_r400_win

more status updates

Change 105982 on 2003/06/13 by bhankins@bhankins_crayola_linux_orl

advance sq-sx control signals by one clock to solve sx timing issues
add support for updated sx hierarchy

Change 105943 on 2003/06/12 by dougd@dougd_r400_linux_marlboro

Added a 2nd write buffer to aluconst, texconst and instruction store to handle
real time writes from cp mixed with non real time writes. This code passes the
mini-regress on tb_sqsp and cp_lcc_tex, cp_lcc_alu, cp_im_load_basic on the gc
testbench but fails cp_lcc_tex_rt and cp_lcc_alu_rt. It appears work for non-realtime.

Added real time prim bit from pix_ctl to ISM in pix_thread_buff when loading a
pixel thread. This bit will allow reading real time constants from the constant stores.

Added VC_wake_up logic.

Change 105924 on 2003/06/12 by vromaker@vromaker_r400_linux_marlboro

timing fixes

Change 105914 on 2003/06/12 by danh@danh_r400_win

Updated r400cl* status.

Change 105891 on 2003/06/12 by rramsey@RRAMSEY_P4_r400_win

more status updates

Change 105889 on 2003/06/12 by danh@danh_crayola1_linux_orl

Changed the "pix: check for buf avail and export count < 16" section of the alu_req generation,
added parentheses around the alloc_size_q & sx_buf_avail logic.

Change 105811 on 2003/06/12 by rramsey@rramsey_crayola_linux_orl

update spsx tracker so msg signal names match the rtl signal names
fix a typo in a pix_rs_input msg

Change 105809 on 2003/06/12 by rramsey@rramsey_crayola_linux_orl

some of the compares had not been updated with the new
vc field in the dump file

Change 105784 on 2003/06/12 by rramsey@rramsey_crayola_linux_orl

fix width of num_params_q

Change 105770 on 2003/06/12 by rramsey@RRAMSEY_P4_r400_win

picking more tests, adding comments to tests with known issues

Change 105750 on 2003/06/12 by smoss@smoss_crayola_linux_orl_regress

removed sq_sp_simd1_instruct_start to coincide with @105565

Change 105592 on 2003/06/11 by llefebvr@llefebvr_r400_linux_marlboro

Added storage element in the SQ to store the valid addresses of the mova so that they can de restored at any instruction that uses the address register. The way it was currently would only work if the use of the address was directly following the MOVA instruction. This fixes r400sq_const_index_02.cpp.

Change 105537 on 2003/06/11 by vromaker@vromaker_r400_linux_marlboro

- added sq_fetch_arb to and removed sq_thread_buff_cntl from system_sq.vcpp
- made a timing fix to gpr alloc

Change 105525 on 2003/06/11 by rramsey@RRAMSEY_P4_r400_win

picking more tests

Change 105465 on 2003/06/10 by vromaker@vromaker_r400_linux_marlboro

- timing fix in pix_thread_buff
- VC interface is connected to vc instruction seq
- TP_SQ_fetch stall replaced by TP_SQ_dec (but not tested at GC level)
- SQ_TP_gpr_wr_addr and SQ_TP_clause removed from top level (and tb updated)
- fetch arbitration for VC and TP updated

- recoded a few lines in gpr alloc to see if it will help timing

Change 105457 on 2003/06/10 by danh@danh_r400_win

Made changes in regards to my simulation results.

Change 105437 on 2003/06/10 by rramsey@RRAMSEY_P4_r400_win

picking some tests to debug

Change 105417 on 2003/06/10 by rramsey@RRAMSEY_P4_r400_win

update status for jun 9 regression

Change 105283 on 2003/06/10 by llefebvr@llefebvr_r400_linux_marlboro

I have added the write enables to qualify the data sent to the SX. This is needed when doing predicated exports or constant waterfalling on exports. This fixed r400sq_const_index_01.cpp test.

Change 105277 on 2003/06/10 by dougd@dougd_r400_linux_marlboro

added output VC_clk_en to sq_rbbm_interface.v and wired it to
SQ_VC_wake_up in sq.v

Change 105052 on 2003/06/09 by smoss@smoss_crayola_linux_orl_regress

a few cadence related changes
1) moved rbbm_event_type to occur after the read of rbbm_re
2) temporarily disabled randomization on the clock for the tb_sqsp dump file

Change 104848 on 2003/06/08 by grayc@grayc_crayola2_linux_orl

fix simd1_valid -> simd1_const_valid

Change 104797 on 2003/06/07 by grayc@grayc_crayola2_linux_orl

add VC ports
modify SP-SQ port names

Change 104715 on 2003/06/06 by danh@danh_r400_win

Updated per simulation results.

Change 104661 on 2003/06/06 by dougd@dougd_r400_linux_marlboro

fixed typo

Change 104616 on 2003/06/06 by llefebvr@llefebvr_r400_linux_marlboro

HW was clamping to 0 on a GPR addressing error. It should clamp to GPR_base of the shader.

Change 104600 on 2003/06/06 by dougd@dougd_r400_linux_marlboro

added missing case value that was causing synopsys to infer latches

Change 104555 on 2003/06/06 by danh@danh_r400_win

Made changes per simulation results.

Change 104554 on 2003/06/06 by dougd@dougd_r400_linux_marlboro

fixed typo -
d_rd0_addr was assigned in two process blocks
and d_rd1_addr was not being assigned at all.

Change 104302 on 2003/06/05 by ashishs@fl_ashishs_r400_win

upadted the script since it was just fetching data till 4000 rows. Now it will fetch data till 10000 rows (break after it finds null rows) and then sort them accordingly....

Change 104261 on 2003/06/05 by rramsey@rramsey_crayola_linux_orl

Fix some wiring issues in tb_sqsp
Add warning msg to tb_sqsp to tell when a test is trying to swizzle
fetch addresses since this is not supported yet in the SP
(didn't make it a failure since some tests are passing with swizzle
-- they must have the same value in all channels)
Fix predicate compare in pix_rs_input tracker
fetch_swizzle bit of instr needed to be muxed based on thread_type
in sqtp tracker

Change 104211 on 2003/06/05 by rramsey@RRAMSEY_P4_r400_win

status from 6_4_2003

Change 104159 on 2003/06/04 by danh@danh_crayola1_linux_orl

Changed count_match[3:0] generation, when param_gen_cycle is high all count_match[3:0] bits will now go high.

Change 104139 on 2003/06/04 by rramsey@rramsey_crayola_linux_orl

turn off debug print for this one too

Change 104076 on 2003/06/04 by dougd@dougd_r400_linux_marlboro

fixed bug in the loading of the write data buffer.

Change 104075 on 2003/06/04 by dclifton@dclifton_r400

added test controller

Change 104046 on 2003/06/04 by smoss@smoss_crayola_linux_orl_regress

removed print statements

Change 104031 on 2003/06/04 by rramsey@rramsey_crayola_linux_orl

Fix trackers so they actually compare, and compare the correct data

Change 104026 on 2003/06/04 by rramsey@RRAMSEY_P4_r400_win

update makefile with spi block, memory changes, etc

Change 103932 on 2003/06/03 by mmantor@mmantor_crayola_linux_orl

update for new pipe disable routing

Change 103931 on 2003/06/03 by danh@danh_r400_win

Updated per simulation results.

Change 103849 on 2003/06/03 by rramsey@rramsey_crayola_linux_orl

Fix a bug in sq_input_arb that was allowing the state machine to go
to IDLE even though a pixel thread was active. This could allow a vtx
and pix thread to try and write into the GPRs at the same time.
Turn tex ctlflow trackers back on in tb_sqsp
Fix TP_SP_data_valid connections in tb_sqsp
Modify alu ctlflow trackers so they can skip over expected instr
with serialize bits set if the rtl does not serialize them

Change 103379 on 2003/05/30 by danh@danh_r400_win

updated per simulation results.

Change 103369 on 2003/05/30 by vromaker@vromaker_r400_linux_marlboro

- fix for width mismatch on thread_id input of vtx TB status regs
- initial pass of VC/TP fetch arbiter (not instantiated in sq.v yet)

Change 103365 on 2003/05/30 by dougd@dougd_r400_linux_marlboro

Added missing wire declaration for param_wrap_0_set

Change 103256 on 2003/05/30 by dougd@dougd_r400_linux_marlboro

fixed bug in wrapping logic for rtn_ptr, read_ptr and stop_ptr for addressing the mapping table address freelist

Change 103204 on 2003/05/29 by dougd@dougd_r400_linux_marlboro

initial submit of a submodule to count and bin pixels for perfmon

Change 103141 on 2003/05/29 by vromaker@vromaker_r400_linux_marlboro

- added simd_num input to the thread buffers (tied low in sq.v) and connected
  it down to the status regs
- added simd_num to the staging registers in the CFS
- connected simd_num thru the target_instr_fetch and tex_instr_queue
  modules (so it is an output of the tex_instr_queue)

Change 103074 on 2003/05/29 by viviana@viviana_crayola2_syn

Added a `include of sq_reg.v for synthesis purposes.

Change 102924 on 2003/05/28 by viviana@viviana_crayola2_syn

Added an additional 48x170 and 16x170 and rebuilt the memories.

Change 102411 on 2003/05/23 by dougd@dougd_r400_linux_marlboro

Simulation only protocol checking logic was moved to a clock process block
to prevent a difference in order of evaluation between vcs and ncverilog
from causing a false error assertion due to a race condition in simulation.

Change 102365 on 2003/05/23 by vromaker@vromaker_r400_linux_marlboro

moved wire declaration of sx_exp_buff_full_0 (and others) before the
instantiation of the status registers to fix ncverilog warning

Change 102264 on 2003/05/23 by vromaker@vromaker_r400_linux_marlboro

- updated pix thread buffer for simd1 (and removed ctl sub module and redundant logic)
- renamed state_read_phase to arb_phase
- fixed CFSM serialize detection (had to add case of fetch initiated by current clause)
- removed reference to sq_thread_buff_cntl in tracker

Change 102193 on 2003/05/22 by danh@danh_r400_win

updated per simulation results.

Change 102154 on 2003/05/22 by rramsey@RRAMSEY_P4_r400_win

Update with 5/17/03 status

Change 102095 on 2003/05/22 by dougd@dougd_r400_linux_marlboro

Added the following new fields to control registers in the rbbm interface:
SQ_CONTEXT_MISC_PERFCOUNTER_REF
SQ_CONTEXT_MISC_YEILD_OPTIMIZE
SQ_FLOW_CONTROL_VC_ARBITRATION_POLICY
SQ_FLOW_CONTROL_SIMD1_DISABLE
SQ_DEBUG_MISC_DB_READ_MEMORY

Change 102052 on 2003/05/22 by danh@danh_crayola1_linux_orl

instr_ptr and instr_ptr_q are now only compared when event_vld_q is low.

Change 102042 on 2003/05/22 by danh@danh_r400_win

updated per simulation results.

Change 102039 on 2003/05/22 by dougd@dougd_r400_linux_marlboro

restored the missing line ".pb_event_state (pb_event_state)," to the instantiation of sq_export_alloc in sq.v that somehow was removed when a merge was done in the last submit

Change 102013 on 2003/05/21 by danh@danh_r400_win

Made changes per the simulations I ran today.

Change 101908 on 2003/05/21 by mmang@mmang_crayola_linux_orl

Fixed bug in waterfalling by grabbing register input of done_bits
instead of registered value when performing init_done_bits operation.

Change 101906 on 2003/05/21 by dougd@dougd_r400_linux_marlboro

added a 2nd read port for VC to texconst and redesigned sq_texconst_wrt_buff to perform opportunistic writes because the write access slot was given up for VC reads

Change 101883 on 2003/05/21 by rramsey@rramsey_crayola_linux_orl

fix pc write addr generation in ais_output
fix cf state machine so unexecuted conditionals don't cause a thread
to end
turn off cf trackers for now
fix a problem in the test bench related to draw pkts with no draw inits
(some cp tests do this)

Change 101881 on 2003/05/21 by danh@danh_crayola1_linux_orl

Changed PB_READ_3 state, it now uses pi_interp_cnt_q instead of interp_cnt_q.

Change 101841 on 2003/05/20 by askende@askende_r400_linux_marlboro

checking in the interpolator control latency changes in SQ and SP.

Change 101642 on 2003/05/19 by vromaker@vromaker_r400_linux_marlboro

- made top level SQ signal changes/additions for SP simd0 and simd1
- added an alu thread arbiter, pairs of alu ctl flow seq, instr
  fetch, instr que, and instr seq modules, and ais_output for simd1
- thread buff cntl sub module removed from vtx thread buffer, and its
  logic moved up to the thread buff level (this still needs to be done
  for the pix thread buffer)
- only one status reg read mux and arb request shifter is needed in the
  thread buffer to support 4 arbiters (since the state mem can only be
  read by one arbiter per cycle), so the duplicates were removed

Change 101575 on 2003/05/19 by smoss@smoss_crayola_linux_orl_regress

changed delay on tp_sp signals

Change 101378 on 2003/05/16 by smoss@smoss_crayola_linux_orl_regress

added field for TP_SP_rf_expand_enable

Change 101314 on 2003/05/16 by moev@moev_r400_linux_marlboro

updates

Change 101168 on 2003/05/15 by rramsey@rramsey_crayola_linux_orl

fix a problem with my param cache allocate fix and fill the hole
in our spsx tracker that let the problem slip through my regressions
(pc write addr was not being checked)

Change 101103 on 2003/05/14 by smoss@smoss_crayola_linux_orl_regress

<Orlando Hardware Regression Results >

Change 101064 on 2003/05/14 by danh@danh_r400_win

Updated fields in regards to my simulation results.

Change 101009 on 2003/05/14 by rramsey@rramsey_crayola_linux_orl

Changes for parameter cache deallocation. Need to multiply dealloc
count by (vs_export_count +1) so the correct number of lines are

freed.

Change 100885 on 2003/05/14 by rramsey@RRAMSEY_P4_r400_win

update validation report

Change 100881 on 2003/05/14 by danh@danh_r400_win

Changed the lines of the simulations that I have run.

Change 100877 on 2003/05/14 by rramsey@rramsey_crayola_linux_orl

Fix 3 issues related to parameter cache allocation/deallocation
1) Move allocate subtract for pc_free_cnt so it happens when
an allocating vtx thread wins arbitration instead of when
the thread is sent to the CFS. This puts the arbitration/
allocate path at four clks (from six) so we can correctly
allocate every four clocks.
2) Deallocs were being dropped in sq_ptr_buff on back to back
row transfers if the first of the pair was the last row
(end of buffer) and the second of the pair had dealloc.
3) Deallocs need to be accumulated in sq_ptr_buff since multiple
row transfers of a pixel vector can be marked with dealloc
and the deallocs are put in the event fifo at end_of_buffer.

Clean up some duplicate code in tb_sqsp and set the default dump
level back to 1 (instead of 3).

Change 100801 on 2003/05/13 by dougd@dougd_r400_linux_marlboro

corrected port width mismatches in sq_aluconst_top; removed unused input and output from sq_const_map_cntl and in it's instantiations in sq_aluconst_top and sq_texconst_top

Change 100795 on 2003/05/13 by dougd@dougd_r400_linux_marlboro

corrected signal names to b1 ports of sq_cfc

Change 100748 on 2003/05/13 by danh@danh_crayola1_linux_orl

instr_ptr and instr_ptr_q are now only compared when event_vld_q is low.

Change 100631 on 2003/05/13 by dougd@dougd_r400_linux_marlboro

Added `define SIMD1 to header.v and corrected connections for SIMD1 in sq.v

Change 100629 on 2003/05/13 by rramsey@rramsey_crayola_linux_orl

Update tb_sqsp for latest SP top level changes
Zero out rbbm fifo data when writing for re_dly

Add a couple of missing wire declarations to sq

Change 100468 on 2003/05/12 by dougd@dougd_r400_linux_marlboro

removed incorrect bit width assignments to eo_rt_aluconst and eo_rt_texconst to prevent compile errors with ncverilog

Change 100453 on 2003/05/12 by rramsey@rramsey_crayola_linux_orl

Update for top level sp changes

Change 100310 on 2003/05/10 by smoss@smoss_crayola_linux_orl_regress

ncsim for sqsp and sx

Change 100167 on 2003/05/09 by rramsey@RRAMSEY_P4_r400_win

Updating status

Change 100164 on 2003/05/09 by dougd@dougd_r400_linux_marlboro

ifdef'd connections in sq.v to sq_aluconst_top.v for the extra SIMD1 memory

Change 100154 on 2003/05/09 by rramsey@rramsey_crayola_linux_orl

Changes for instruction store addressing (wrapping and absolute)
Add absolute addressing for cf and exec addresses to cfs
Add wrapping for jumps and calls to cfs
Add wrapping for execute addresses to cfs
Fix wrapping in instr_fetch (vtx wrap at pix_base-1)

These changes fix cp_event_timestamp_instruction_loading_stall at tb_sqsp

Change 100118 on 2003/05/09 by dougd@dougd_r400_linux_marlboro

added 2nd memory to sq_cfc to support SIMD1 and ifdef'd the connections in sq_cfc and sq.v

Change 100015 on 2003/05/08 by mmantor@mmantor_crayola_linux_orl

<sq_ais_output - re-ordered kill_mask going to the sx so bits flow in order msb->lsg sp2(v3-v0)sp0(v3-v0)) to match exp_mask
- removed improper final update of kill mask with predication mask
- enable export_mask for all exports
SX_PA_interfaces.v - fixed checker for back to back transfers
SX_RB_interfaces.v - hooked up to 7 bit sx_rb_index and rb_sx_index instead of incorrect 8 bits
sx.v - changed interfaces for sx_rb and rb_sx interfaces to become 7 bits instead of 8 bits

tb_sx.v - changed sx inputs to be 7 bits instead of 8 bits on the above index interfaces
tbmod_fake_sp.v - reordered the kill mask and enabled channel mask for exports
sx_export_buffers.v - moved register after export mems and only load when memory read, mimized client read muxes added input rotate muxes for export to memory operations and indivual write address for each memory and set up predication, kill_mask, alpha kill,and channel mask in the determination of writing data into the export buffers
sx_export_control.v - removed dead clock on rb and pa data fetch interface and client and made arbiter behave as round robin and removed unecessary second input register, added support for z render targets and multiple render targets and clean up items
ex_export_alloc_dealloc.v - enabled channel mask, kill mask, export_mask, and apha test conditioning of valid bitsa doubled the free rate>

Change 99918 on 2003/05/08 by dougd@dougd_r400_linux_marlboro

fixed typo

Change 99912 on 2003/05/08 by dougd@dougd_r400_linux_marlboro

doubled the instruction store memory, changed the access allocation to accomdate SIMD1 and VC, and `ifdef'd the connections for SIMD1 in sq.v

Change 99520 on 2003/05/07 by mmang@mmang_crayola_linux_orl

Bug occurred where first_in_clause was getting lost when instr_queue
was full. Previously, internal first_in_clause register was cleared
with tif_rts. Had to delay clearing to tif_rts & tiq_rtr.

Change 99346 on 2003/05/06 by mmang@mmang_crayola_linux_orl

Fixed bug (I created) related to initializing the constant address
register valids at the beginning of a clause. I used ais_init_pred
which in some cases was too late. Created new ais_init_const_addr
that is 3 clocks sooner.

Change 99315 on 2003/05/06 by vromaker@vromaker_r400_linux_marlboro

fixed typos that were causing cp_e2polyscanlines_simple to fail

Change 99123 on 2003/05/05 by rramsey@rramsey_crayola_linux_orl

Add some control to hold off inputs at vs/ps done events
Increase utb_tp_req_fifo depth
Change writes into vtx/pix done fifos so they only happen on the first
draw_init for a context

Change 99043 on 2003/05/05 by vromaker@vromaker_r400_linux_marlboro

- added VC ctl flow seq, instr fetch, instr que and instr seq, and top level IOs
- made some leda fixes

- added non time multiplexed gpr write address output to VC and TP (gpr_dst_addr[6:0])

Change 99041 on 2003/05/05 by rramsey@RRAMSEY_P4_r400_win

Regression results from 5/4/03
3451 tests: 66% Pass, 12% Fail, 23% incomplete

Change 98861 on 2003/05/02 by smoss@smoss_crayola_linux_orl_regress

more sq stuff

Change 98818 on 2003/05/02 by smoss@smoss_crayola_linux_orl_regress

added missing dump

Change 98793 on 2003/05/02 by rramsey@rramsey_crayola_linux_orl

Check in Dan's fixes for the control flow trackers
Turn internal trackers back on in tb_sqsp

Change 98773 on 2003/05/02 by mmang@mmang_crayola_linux_orl

1. Added constant address register valids to validate the
address register data. The valid is set when address register
is written. If valid is not set, sequencer will not waterfall
those vertices or pixels. This disables waterfalling for
predicated off writes and improperly initialized contant
address registers.
2. Fixed bug in sqs_alu_instr_seq for phase 3 snooping of
constant address registers bus. Previously, this snooping
did not account for predication of those registers.
3. Fixed bug where ais_load_done_bits was not hooked up. This
signal disables previous vector/scalar management which needs
to be turned off during constant waterfalling. With bug,
pvps logic went unknown which caused unknowns to eventually
propagate in and out of the gprs.
4. Fixed bug where non-optimized offset was not being determined
properly. non_opt_offset is determined by a priority encoder
of p0_done, p1_done, p2_done, and p3_done.
5. With advent of constant address register valids, created
waterfall_active_q to properly init and avoid re-initing of
different pixel and vertex done bits.

Change 98750 on 2003/05/02 by viviana@viviana_crayola2_syn

Memory increased from 48x155 to 48x170.

Change 98577 on 2003/05/01 by smoss@smoss_crayola_linux_orl_regress

reverting changes due to over-engineered process

Change 98571 on 2003/05/01 by smoss@smoss_crayola_linux_orl_regress

sometime it helps when you save the file first

Change 98569 on 2003/05/01 by smoss@smoss_crayola_linux_orl_regress

added FSDB_DUMP option for VCS

Change 98509 on 2003/05/01 by smoss@smoss_crayola_linux_orl_regress

removed tb_sqsp

Change 98462 on 2003/05/01 by vromaker@vromaker_r400_linux_marlboro

- added bits and re-arranged the order of bits in the status register
- added VC support in thread buffers (vc request from status register,
read muxes, connections to other modules, etc.)
- removed is_subphase and made is_phase 3 bits
- removed cfc_phase
- expanded state_read_phase to 2 bits
- changed the strapping and phase relationships on the ctl flow seqs
- SQ_SP_fetch_swizzle and SQ_SP_fetch_resource outputs added
- disabled internal SQ trackers and changed to DEBUG_PRINT ifdef in tb_sqsp.v

Change 98398 on 2003/04/30 by smoss@smoss_crayola_linux_orl_regress

new sq stuff

Change 98397 on 2003/04/30 by grayc@grayc_crayola2_linux_orl

new tb

Change 98367 on 2003/04/30 by rramsey@rramsey_crayola_linux_orl

these trackers were looking at the wrong register stage to determine
thread_id and thread_type

Change 98343 on 2003/04/30 by ashishs@fl_ashishs_r400_win

Correcting an error from script since it wasn't updating the user's comments and locked by user correctly. Also adding an empty XLS file which is used by the script to add and merge data

Change 98307 on 2003/04/30 by ashishs@fl_ashishs_r400_win

fixed a small error in the script because of which it wasnt getting the comments from the report. Also updated some comments.

Change 98283 on 2003/04/30 by viviana@viviana_crayola2_syn

Files no longer used in the SQ.

Change 98274 on 2003/04/30 by rramsey@rramsey_crayola_linux_orl

change if(`DEBUG_PRINT) to `ifdef DEBUG_PRINT so trackers
work at gc level

Change 98261 on 2003/04/30 by ashishs@fl_ashishs_r400_win

added the script for updating the XLS hadware regression data. Can have more enhancements depending on requirements

Change 98144 on 2003/04/29 by rramsey@rramsey_crayola_linux_orl

Add internal trackers to tb_sqsp, clean up memory files listed in tb.f
Remove DEBUG_PRINT from tb.f, it should be specified in vcsopts.f

Change 98142 on 2003/04/29 by rramsey@rramsey_crayola_linux_orl

timing fix for rbi_addr

Change 98140 on 2003/04/29 by rramsey@rramsey_crayola_linux_orl

fix a typo in a signal path

Change 98132 on 2003/04/29 by rramsey@rramsey_crayola_linux_orl

update trackers for new fields in dump files and make them
work for events

Change 98079 on 2003/04/29 by rramsey@rramsey_crayola_linux_orl

Fix a bug with alu1's trigger
Add define control for comment printing

Change 98067 on 2003/04/29 by danh@danh_crayola_linux_orl

Made type_serialize_1 and vc_request_1 changes.

Change 97992 on 2003/04/28 by dougd@dougd_r400_linux_marlboro

fixed some Leda reported problems

Change 97991 on 2003/04/28 by dougd@dougd_r400_linux_marlboro

added R500 dual read ports and extra memories.

Change 97962 on 2003/04/28 by danh@danh_crayola_linux_orl

Made signal changes in regards to .dmp file changes.

Change 97961 on 2003/04/28 by danh@danh_crayola_linux_orl

Made signal name changes in regards to .dmp file changes.

Change 97958 on 2003/04/28 by danh@danh_crayola_linux_orl

Made signal changes in regards to the .dmp file changes.

Change 97956 on 2003/04/28 by danh@danh_crayola_linux_orl

Added jump_call_addr registers.

Change 97892 on 2003/04/28 by danh@danh_crayola_linux_orl

no changes made.

Change 97732 on 2003/04/25 by danh@danh_crayola_linux_orl

Changed signal names per sq_pix_control_flow_alu.dmp

Change 97708 on 2003/04/25 by rramsey@rramsey_crayola_linux_orl

Move inc for event thread count to front of event fifo
They were still happening on the same clk as real threads

Change 97670 on 2003/04/25 by rramsey@rramsey_crayola_linux_orl

Change buildtb and buildkdb to use tb.f for libraries and compile options
to keep from having to add files in two places
Couple of bug fixes/enhancements for tb_sqsp
Fix path define for sp_macc tracker when running tb_sqsp

Change 97538 on 2003/04/24 by ygiang@ygiang_r400_pv2_marlboro

added: more sq perf counters

Change 97402 on 2003/04/24 by kmeekins@kmeekins_crayola_linux_orl

Initial release.
Tracker used to test the inputs and outputs of all MACC units within the shader pipes.

Change 97152 on 2003/04/23 by dougd@dougd_r400_linux_marlboro

added logic to control vtx perf counters to sq_vtx_ctl.v and sq.v; fixed bug in write logic in sq_aluconst_wrt_buf.v

Change 96990 on 2003/04/22 by viviana@viviana_crayola2_syn

Ran cover on the sq_rf.cnt to add the new 16x170 and 48x170 memories.

Change 96981 on 2003/04/22 by viviana@viviana_crayola2_syn

Added TST_awt_enable to the interfaces with ss/sq_pix_thread_buff.v and
ss/sq_vtx_thread_buff.v.
Replaced the 16x155 and 48x155 memories with 16x170 and 48x170 respectively.
Replaced the memory to be compiled in buildtb from the 155 to the 170.

Change 96948 on 2003/04/22 by viviana@viviana_crayola2_syn

Changed the name of the FIFO.

Change 96947 on 2003/04/22 by viviana@viviana_crayola2_syn

Removed width from paramenter definitions.

Change 96946 on 2003/04/22 by viviana@viviana_crayola2_syn

Added done_vector to sensitivity list at line 902.
Removed `SQ_SRCB_PHASE from sensitivity list at line 1018.
Added isr_thread_type_q to sensitivity list at line 1233.

Change 96876 on 2003/04/22 by rramsey@rramsey_crayola_linux_orl

only compare if one of the vector unit bits is valid

Change 96738 on 2003/04/21 by mmang@mmang_crayola_linux_orl

Fixed bug in sq_ais_output.v related to address register write and
predication. Fixed a variety of tests to not use uninitialized gpr
or address registers. 2 tests still fail because of previous vector
scalar swizzle bug, 1 test still fails because of MOVA hardware bug, and
1 test still fails because of predicated address register write causes
XXXXXX which causes waterfalling to hang.

Change 96623 on 2003/04/21 by bhankins@bhankins_crayola_linux_orl

add support for including SX units into tb_sqsp.v

Change 96455 on 2003/04/18 by bhankins@bhankins_crayola_linux_orl

initial checkin to optionally include (not included by default) two SX units
with associated support logic and trackers.

Change 96445 on 2003/04/18 by rramsey@rramsey_crayola_linux_orl

Move compares into a task, add a flag
to enable marking x vs 0 compares as warnings

Change 96389 on 2003/04/18 by mzini@mzini_crayola_linux_orl

Temporarily removed the checking of control bits until the hardware catches up

Change 96318 on 2003/04/17 by rramsey@rramsey_crayola_linux_orl

change Openning to Opening

Change 96278 on 2003/04/17 by scamlin@scamlin_crayola_unix_orl

.

Change 96252 on 2003/04/17 by rramsey@rramsey_crayola_linux_orl

Add new signal between sq_vtx_ctl and sq_input_arb to mark gpr_ld_busy
which fixes a deadlock condition where gpr_ld state machine is waiting
on an ack from the arb, which is waiting on vsr_ld machine not busy, which
is waiting on the gpr_ld machine to finish.
Add a reset for sp_sel in sq_vtx_ctl when the vsr_ld machine goes from ld
to idle. This is needed if there is valid data sitting in the vgt fifo since
a vsr load will happen on the following clock.
Connect pred_kill_valid bits in tb_sqsp.
Fix typo in visr_wr tracker.

Change 95859 on 2003/04/16 by rramsey@rramsey_crayola_linux_orl

fix width

Change 95847 on 2003/04/16 by rramsey@rramsey_crayola_linux_orl

script to build the verdi kdb for tb_sqsp

Change 95839 on 2003/04/16 by rramsey@rramsey_crayola_linux_orl

Multiple changes throughout the SQ to get tb_sqsp working, and to fix bugs
uncovered by the new testbench
sq_gpr_alloc - wrap was broken for vertex side if ptr wrapped exactly at max
sq_rbbm_interface - context/new_ld sent to constant mems was being pulled from
wrong
side of skid buffer
sq_pix_thread_buff - change to send context_done event back to CP on the pixel side
buildtb, tb_sqsp, tbtrk_* - changes for new sp/spi configuration and new dump file
fields

Change 95810 on 2003/04/16 by dougd@dougd_r400_linux_marlboro

fixed bug in the pix_cntx counter increment signal where both event and pix were trying to increment during the same cycle

Change 95623 on 2003/04/15 by mzini@mzini_crayola_linux_orl

New SQ trackers

Change 95621 on 2003/04/15 by mzini@mzini_crayola_linux_orl

Using an unregistered data-ready signal now to trigger compare

Change 95490 on 2003/04/14 by vromaker@vromaker_r400_linux_marlboro

fix for CFSM end-of-clause detection

Change 95457 on 2003/04/14 by danh@danh_crayola_linux_orl

Initial Release.

Change 95404 on 2003/04/14 by rramsey@rramsey_crayola_linux_orl

Temporary fix to let flush events flow through the SC if they are not
preceded by a draw_init.
Note flushes are still broken if you want to use them for syncing state changes,
for example changing bad_pipe.

Change 95391 on 2003/04/14 by rramsey@rramsey_crayola_linux_orl

Change SQ_DEBUSSY option to control only SQ dumping
Add SP_DEBUSSY option to control dumping of SPs
Add some internal trackers
Fix width on tp_sqsp_thread_id

Change 95174 on 2003/04/11 by hartogs@fl_hartogs

Added hooked up new "thread_type" signal from SQ module. Enhanced timeout detection.

Change 94999 on 2003/04/10 by ygiang@ygiang_r400_pv2_marlboro

fixed: sq perf counter "sq_vertex_vectors_sub"

Change 94873 on 2003/04/10 by askende@askende_r400_linux_marlboro

releasing the following changes:
1. creation of the new SPI block

2. top level changes to support 8 SP instances
3. tracker changes to support a few IO name changes

Change 94834 on 2003/04/09 by mmantor@mmantor_crayola_linux_orl

<fixed a problem in ptr_buff where he hangs on an event that is not a pixel event>

Change 94830 on 2003/04/09 by mmantor@mmantor_crayola_linux_orl

<SQ/SX/SP out of order thread completion and remove redundant storage in sp for sq/sx communitcations some sq cfs bug fixed and texture kill mask generation and other misc things>

Change 94724 on 2003/04/09 by mzini@mzini_crayola_linux_orl

New SQ trackers

Change 94718 on 2003/04/09 by viviana@viviana_crayola_linux_orl

SQ trackers for the sequencer control flow instructions.

Change 94518 on 2003/04/08 by vromaker@vromaker_r400_linux_marlboro

fix for bug caused by resource change between EXEC control flow
instructions

Change 94505 on 2003/04/08 by rramsey@rramsey_crayola_linux_orl

don't inc vtx thread counter if thread is an event

Change 94211 on 2003/04/07 by hartogs@fl_hartogs

Template Debussy waveform file.

Change 93959 on 2003/04/04 by vromaker@vromaker_r400_linux_marlboro

temporarily disabled PVPS src select swizzle because it was causing SP tests
from Andi's mini_regress to fail

Change 93788 on 2003/04/03 by dougd@dougd_r400_linux_marlboro

added event_context register to correctly capture context when sending a vtx event to the thread buffer.

Change 93640 on 2003/04/03 by hartogs@fl_hartogs

Added include file "vgt_reg.v" to prevent compiler errors during Modelsim compile.
Fixed wire definition to match output port definition.

Change 93585 on 2003/04/03 by dougd@dougd_r400_linux_marlboro

move event filters to inputs of SQ in ptr_buff and vtx_ctl and remove from thread_buff_cntl, modify cntx0-17 busy counters in ptr_buff to use new event logic, add RST_VTX_CNT functionality to sq_vtx_ctl, add SQ_CP_event and SQ_RB event functionality to pix_thread_buff and vtx_thread_buf, remove obsolete SQ_CP_event functionality from thread_buff_cntl.

Change 93489 on 2003/04/02 by vromaker@vromaker_r400_linux_marlboro

added end of clause detection for serialization and resource change to CFSM;
fix for first_in_clause related to PVPS detection;
added SQ_SP_thread_type and SQ_TP_thread_type outputs to sq.v;
added predicate to Tex IQ - now predicate goes from TCFS, thru TIF, thru TIQ, to the TIS;
removed q1 pipeline stage for SP predicate data in AIS Output;

Change 93099 on 2003/04/01 by hartogs@fl_hartogs

Fixed logic that holds off SC injector.
Fixed bug in threaded empty signal for tp_sqsp_dmp_not_empty.
Hacked logic that frees state contexts so that it sorta works.

Change 93054 on 2003/04/01 by rramsey@rramsey_crayola_linux_orl

PV/PS determinations need to be made on post-swizzled component selects

Change 93053 on 2003/04/01 by dougd@dougd_r400_linux_marlboro

added context_id or state to event info stored in the status_reg to correct a bug in the state logic supporting cntx0 and cntx17 busy

Change 93026 on 2003/03/31 by grayc@grayc_crayola_linux_orl

minor changes for gc testbench

Change 92970 on 2003/03/31 by dougd@dougd_r400_linux_marlboro

added pix event_id to qualify events to increment the cntx17 busy counter. Also added `include "vgt_reg.v" and removed hard coded parameters.

Change 92904 on 2003/03/31 by hartogs@fl_hartogs

Added TP_SP latency controls requested by Mantor.

Change 92884 on 2003/03/31 by hartogs@fl_hartogs

Cleaned-up some "TODO" items.

Change 92679 on 2003/03/28 by hartogs@fl_hartogs

Hooked-up new thread-id ports on SQ. Testbench should now handle most tests.

Change 92451 on 2003/03/27 by hartogs@fl_hartogs

Added several missing input ports to sub-modules.

Change 92428 on 2003/03/27 by hartogs@fl_hartogs

Deleted reduntant wire declaration that was causing an error in modelsim.
Added explicit declarations for a bunch of implicitly used wires. The lack of the explicit declarations was causing warnings in Modelsim.

Change 92324 on 2003/03/27 by dougd@dougd_r400_linux_marlboro

fixed bug with address wrapping in the gpr_wr_addr generation for pix and vtx

Change 92303 on 2003/03/27 by mmantor@mmantor_crayola_linux_orl

<export_id expanded from one to four bit, end_of_clause added to CFS-TIF interface and last_in_clause flag passed down with instruction to eventually trigger a free_done. thread_id outputs added to the SQ_SP and SQ_TP interfaces for testbench, and added sq_sx control signals for exp_table read >

Change 92262 on 2003/03/26 by hartogs@fl_hartogs

Added untested code for random backpressure on the SQ_TP interface.
Added untested code for random starve pressure on the TP_SP interface.

Change 92142 on 2003/03/25 by hartogs@fl_hartogs

This version passes milestone_tri. Trackers and injectors are ready for multi-threading when those signals become available.

Change 92139 on 2003/03/25 by hartogs@fl_hartogs

This version passes the "milestone_tri" test completely. First working version for texture fetch.

Change 91978 on 2003/03/25 by hartogs@fl_hartogs

Incremental check-in. This version will run "milestone_tri" to completion (including injecting the texture fetch data); however the simulation mismatches on the SPSX tracker.

Change 91977 on 2003/03/25 by hartogs@fl_hartogs

Changed "check flags" on one of the SQ memories to minimize garbage output during simulation.

Change 91976 on 2003/03/25 by hartogs@fl_hartogs

Changed dump file name from tp_sq.dmp to tp_sqsp.dmp.

Change 91763 on 2003/03/24 by hartogs@fl_hartogs

Qualified use of "q_skid_rbbm_a" with "!q_skid_empty". This was done to avoid X's on "sel_gfx_vgt_draw_initiator" which corrupted a counter "map_copy_cntr". The corruption of "map_copy_cntr" could only be corrected by reset.

Change 91754 on 2003/03/24 by rramsey@rramsey_crayola_linux_orl

Fix for gpr write tracker

Change 91569 on 2003/03/21 by dougd@dougd_r400_linux_marlboro

added Real Time input to each of the constant store to enable the correct addressing of RT constants from the SQ

Change 91422 on 2003/03/20 by dougd@dougd_r400_linux_marlboro

fixed bugs in the rbi_rd_state machine

Change 91219 on 2003/03/20 by dougd@dougd_r400_linux_marlboro

fix bug in real time write address logic

Change 91126 on 2003/03/19 by dougd@dougd_r400_linux_marlboro

fixed bug in incrementing pix_cntx17_cnt with back to back events

Change 90960 on 2003/03/19 by smoss@smoss_crayola_linux_orl

added new sq trackers

Change 90915 on 2003/03/19 by dougd@dougd_r400_linux_marlboro

added I/O TEST PORTS for bist and scan

Change 90861 on 2003/03/19 by rramsey@rramsey_crayola_linux_orl

Changes to sp_sel and valid logic to get bad_pipe working

Change 90773 on 2003/03/18 by dougd@dougd_r400_linux_marlboro

added logic to drive SQ_CNTX0_BUSY, SQ_CNTX17_BUSY. This change should complete this functionality.

Change 90733 on 2003/03/18 by dougd@dougd_r400_linux_marlboro

added counters and control for pix cntx0, cntx17 busy

Change 90622 on 2003/03/17 by hartogs@fl_hartogs_linux

First crack at VCS build script for tb_sqsp

Change 90551 on 2003/03/17 by dougd@dougd_r400_linux_marlboro

added code in sq_status_reg to decement the thread counters in sq_vtx_cntl used for SQ_CNTX0_busy, SQ_CNTX17_busy

Change 90313 on 2003/03/14 by hartogs@fl_hartogs

Added multi-threaded code to tbtrk_sqtp.v. (Thread id and type are hard-coded to 0 and 1, respectively, until these signals are available from the sq.)
Fixed port name typos in tbtrk_spsx.v.

Change 90002 on 2003/03/13 by viviana@viviana_crayola_linux_orl

Corrected some signal paths to work at the gc level testench.

Change 89954 on 2003/03/13 by viviana@viviana_crayola_linux_orl

Tracker to test the interface between the Vertex Input Control of the sq and the
VSR's of the SP, during a write.

Change 89810 on 2003/03/12 by hartogs@fl_hartogs

Minor corrections to tbtrk_sqtp.
Modularized the SP/SX tracker.

Change 89746 on 2003/03/12 by hartogs@fl_hartogs

Modularized the SQTP tracker (per Chris Gray's request).

Change 89740 on 2003/03/12 by hartogs@fl_hartogs

Somehow this one slipped through the cracks. This one should have been changed when the "pc_free_cnt_q" signal width
was increased from 7 to 8 bits.

Change 89588 on 2003/03/11 by hartogs@fl_hartogs

Added tp_sq.dmp to list.

Change 89538 on 2003/03/11 by hartogs@fl_hartogs

Interim Check-in

Change 89518 on 2003/03/11 by hartogs@fl_hartogs

Dummy files currently uses by tb_sqsp testbench.

Change 89516 on 2003/03/11 by hartogs@fl_hartogs

Interim check-in.

Change 89515 on 2003/03/11 by hartogs@fl_hartogs

Changed "pc_free_cnt_q" to 8 bits so that it could represent the maximum free count of 128. Propagated this change to
ss/sq_status_reg.v and to ss/sq_vtx_thread_buff.v. Added ifdef SIM code to check for overflow and underflow of this
counter.
Added condition for simultaneous occurrance of pc_alloc and pb_dealloc_vld.

Change 89474 on 2003/03/11 by vromaker@vromaker_r400_linux_marlboro

change to make texture requests wait on alu_instr_pending

Change 89448 on 2003/03/10 by mmantor@mmantor_crayola_linux_orl

<1. Added timestamp to dum_mem read and write from same location error message. 2. Moved flat/gouroud shading and provoking vertex to sq-pc from the sx and worked on ptr instead of data 3.Added control for the texture cylinderical wrapsubcycling. 4. Add rt parameter cache ptr selection in sq 5.Clamped and wrapped pc_ptrs in sq 6. Added support for points and lines in the parameter cache ptr determination. 7. prep seperate write address for export to memory 8. tmp fix for deallocation of export memory deallocation. 9. remove some old comment out code and redundant logic >

Change 89039 on 2003/03/07 by dougd@dougd_r400_linux_marlboro

added `include "register_addr.v"

Change 88929 on 2003/03/06 by vromaker@vromaker_r400_linux_marlboro

fix to CFS that prevents a new thread from entering when the thread ID
in the input pipe stage is different than the thread ID in the output pipe stage;
also changed triangle size to 150 for sq_tests test case pred_eq_vec

Change 88816 on 2003/03/06 by dougd@dougd_r400_linux_marlboro

added ports to support cntx0_busy, cntx17_busy

Change 88639 on 2003/03/05 by vromaker@vromaker_r400_linux_marlboro

a few minor updates, mostly comment related; added q2 verison of

state_head_ptr for use in state read addr calcualtion in vtx thread buff

Change 88552 on 2003/03/05 by dougd@dougd_r400_linux_marlboro

needed to subtract RT base address (`SQ_FETCH_RT_0) from incoming address for RT writes and reads.

Change 88512 on 2003/03/05 by rramsey@rramsey_crayola_linux_orl

Fix a bug with the valid bit inits that was causing tri128_pix4 to fail
Change the vsr_ld machine to alternate between buf0 and buf1 so pattern
is deterministic and can be compared vs emulator

Change 88400 on 2003/03/04 by vromaker@vromaker_r400_linux_marlboro

fix for dealloc_space width; status register and thread buffer updates for
status register writes; status register fix for clearing the event_valid (and all
other bits) on a pop; new_thread flag now generated in the instr fetch module and
send dowm thru the AIQ; fix for the setting of thread_valid status

Change 88117 on 2003/03/03 by hartogs@fl_hartogs

Added dum_mem_p2 model back into the code with USE_BEHAVE_MEM compiler directive.

Change 87997 on 2003/03/03 by dougd@dougd_r400_linux_marlboro

missing term in eqn for skid_re_hold for tex_rt_rd caused tex_rt_rd_req to assert for only 1 cycle and not wait for the data ack

Change 87726 on 2003/02/28 by vromaker@vromaker_r400_linux_marlboro

another merge fix

Change 87675 on 2003/02/28 by rramsey@rramsey_crayola_linux_orl

Change cf machine to use program_base derived off of isr, and then
register that value on load_osr for sending to the tip

Change 87632 on 2003/02/28 by vromaker@vromaker_r400_linux_marlboro

another merge fix

Change 87622 on 2003/02/28 by vromaker@vromaker_r400_linux_marlboro

bad merge - retry...

Change 87620 on 2003/02/28 by vromaker@vromaker_r400_linux_marlboro

MOVA related change - due to the extra cycle required by SP for timing,
had to conditionally mux the interface register on the last phase of the address
load into the constant waterfalling logic (to be used instead of last quarter
of the address register since the start of the CWF logic now overlaps the
address register load by one cycle)

Change 87440 on 2003/02/27 by mmang@mmang_crayola_linux_orl

Predicated parameter cache writes.

Change 87408 on 2003/02/27 by vromaker@vromaker_r400_linux_marlboro

Change 87398 on 2003/02/27 by donaldl@donaldl_crayola_linux_orl

Created early version of pix_winner_q going from sq_alu_thread_arb to
sq_pix_thread_buff in order to create a registered version of alu_winner_final.
Done to reduce critical path of the allocation_available signals in
sq_exp_alloc_ctrl.v

Change 87269 on 2003/02/27 by donaldl@donaldl_crayola_linux_orl

Removed allocation of 48 locations for vtx pass thru

Change 87207 on 2003/02/26 by hartogs@fl_hartogs

Added comments that question a few lines of code.
Added ifdef SIM check for overflow and underflow on the the pc_free_cnt_q signal.

Change 87206 on 2003/02/26 by hartogs@fl_hartogs

Changed dealloc_cnt signal going into the event fifo so that it is masked-out (zeroed) for two_clock_xfer that is not end_of_buiffer. This change was made because I observed the logic push the new signal into the event fifo with the dealloc count, and then pushing the dealloc count in again with the pix_vector_valid signal later.
Changed ef_pop signal so that it will not pop a non-zero dealloc unless the new bit is not set. This change was made because the absence of the change above allowed a new signal to go in with a non-zero dealloc_cnt which was then permaturely popped instead of waiting for the vertex vector to be done.
Added some TODO comments for some hardcoded parameters that should come from autoreg include files.
Added some ifdef SIM code that checks for under flow of vtx_sync_cnt_q (which was an observable result of the problems above).

Change 87184 on 2003/02/26 by dougd@dougd_r400_linux_marlboro

when vgt_end_of_vector occurs on the 1st and only data, the data sent to the SP is delayed by 1 cycle but the vsr_wrt_addr was not. Fixed

Change 86670 on 2003/02/25 by vromaker@vromaker_r400_linux_marlboro

fixed gpr address calculation for CONST scalar opcodes

Change 86509 on 2003/02/24 by dougd@dougd_r400_linux_marlboro

corrected error introduced in the last version

Change 86412 on 2003/02/24 by dougd@dougd_r400_linux_marlboro

fixed bit width mismatch in the wrt addr assignment when doing RT

Change 86136 on 2003/02/21 by dougd@dougd_r400_linux_marlboro

fixed bug in o_vector_valid - it counted a single vert as two when there was only one vert in the vector and end_of_vector was also asserted.

Change 86121 on 2003/02/21 by vromaker@vromaker_r400_linux_marlboro

changed context_id (state) used for CFC reads to be that from the input
pipeline register

Change 86092 on 2003/02/21 by vromaker@vromaker_r400_linux_marlboro

added code to disable PVPS detection on 2nd thru last iterations of a
const waterfall loop; added last_in_shader output from CFS that is separate
from the cfs_last_instr status bit that is sent back to the thread buffer

Change 85975 on 2003/02/21 by hartogs@fl_hartogs

The version has code in the testench to track the SX buffer availability and to free buffers after
the SP has exported the data to the SX. It also has the SQ/SP instruction interface time de-multiplexed
and split out by field.

Change 85737 on 2003/02/20 by donaldl@donaldl_crayola_linux_orl

Fixed alu_req equation for handling last_instr_q and first_thread_q.
Changed nxt_pix_last_alloc counter to nxt_pix_last counter.

Change 85660 on 2003/02/20 by vromaker@vromaker_r400_linux_marlboro

updated fifo ctl to output a registered count, and changed the ctl
logic to use the counter for full, empty, etc.

Change 85659 on 2003/02/20 by vromaker@vromaker_r400_linux_marlboro

now enable PVPS detection only on 2nd to last consective instructions of a thread; also added more support for MUL_CONST (force src_c_sel.x to GPR)

Change 85653 on 2003/02/20 by vromaker@vromaker_r400_linux_marlboro

fixed so that only vertex valid bits are swapped (pixel valid bits are
swapped on input from SC)

Change 85650 on 2003/02/20 by vromaker@vromaker_r400_linux_marlboro

fixed thread changed detection logic - the first_in_group and last_in_group
outputs now mark the start and end of an uninterrupted stream of target instructions
from the same thread

Change 85640 on 2003/02/20 by vromaker@vromaker_r400_linux_marlboro

reorderd LOD correction bits to match valid_bits

Change 85595 on 2003/02/20 by scamlin@scamlin_crayola_unix_orl

added testreg

Change 85525 on 2003/02/20 by scamlin@scamlin_crayola_unix_orl

change test port name

Change 85487 on 2003/02/20 by dougd@dougd_r400_linux_marlboro

fixed bug in "double mode" vsr address generation. Also added logic to choose correct SP from the IDLE state when some SP's have been disabled by the BAD_SP bits.

Change 85405 on 2003/02/19 by dougd@dougd_r400_linux_marlboro

modified system_sq.vcpp and vcs.ini to find the new virage hs memories. Backed out the new hs ram in texconst because it doesn't work.

Change 85398 on 2003/02/19 by pmitchel@pmitchel_r400_laptop

recovering deleted file

Change 85337 on 2003/02/19 by scamlin@scamlin_crayola_unix_orl

delete these files and use the _rtl.v versions

Change 85336 on 2003/02/19 by scamlin@scamlin_crayola_unix_orl

new virage hs memories
modified default parameter values to work around a synthesis hang

Change 85215 on 2003/02/19 by donaldl@donaldl_crayola_linux_orl

Added nxt_pix_alloc and nxt_pix_last_alloc counters.

Change 85031 on 2003/02/18 by vromaker@vromaker_r400_linux_marlboro

'last' optimization within EXEC_END added; also fixed last so that without
optimization, it will be set on the last target instruction in the EXEC_END (was
setting it on the first instruction of an EXEC_END, which for the majority of
our tests is the same as the last instruction of an EXEC_END)

Change 84707 on 2003/02/16 by vromaker@vromaker_r400_linux_marlboro

the EXSM was draining valid instructions from the exec ppb on a thread buffer update - fixed by
adding the thread id to the ppb, then checking it to see if it is the same as
the thread that's being updated before removing it (if the thread id is
different then it is not removed and the drain is complete)

Change 84689 on 2003/02/16 by vromaker@vromaker_r400_linux_marlboro

another CFS fix - this time for subroutine calls, but in general it should affect all flow control instructions

Change 84625 on 2003/02/15 by vromaker@vromaker_r400_linux_marlboro

fix for loops - was using nest_level from the register that was changed to the output pipe stage

Change 84577 on 2003/02/14 by ygiang@ygiang_r400_pv2_marlboro

added: more sq performance counters

Change 84449 on 2003/02/14 by mmang@mmang_crayola_linux_orl

Made init_pred dependent upon new_thread instead of first_in_group.

Change 84438 on 2003/02/14 by dougd@dougd_r400_linux_marlboro

changed logic for exp_buf_empty to used only the buffer counts and not their control (update) signals. This signal is used to turn off the clocks for the SP,SX and TP.

Change 84436 on 2003/02/14 by dougd@dougd_r400_linux_marlboro

added logic and wires up to sq to provide events to trigger the perfmon counters

Change 84405 on 2003/02/14 by vromaker@vromaker_r400_linux_marlboro

minor updates: some comments added/removed; also removed winner_ack input from status reg

Change 84304 on 2003/02/13 by vromaker@vromaker_r400_linux_marlboro

removed some commented out code from ais_output; added a pipeline stage to the ctl_flow_seq and decoupled the CFS and EXEC state machines - the arbiters now issue a thread every four cycles

Change 84229 on 2003/02/13 by ygiang@ygiang_r400_pv2_marlboro

added: more sq perf counters

Change 84111 on 2003/02/13 by dougd@dougd_r400_linux_marlboro

fixed a bug in the VGT_SQ interface to allow both _indx_valid and _end_of_vector on the first and only data transfer of a vector. Also added a simulation only protocol monitor to detect _end_of_vector when no data at all had been sent for the vector.

Change 83696 on 2003/02/11 by ygiang@ygiang_r400_pv2_marlboro

added: more performance counters for sq

Change 83645 on 2003/02/11 by hartogs@fl_hartogs

Update. Testbench can run non-fetch tests.

Change 83482 on 2003/02/11 by dougd@dougd_r400_linux_marlboro

changed sq_vtx_thread_buffer to use the status_reg at state_head_ptr_q for events instead of always using status_data_0

Change 83434 on 2003/02/10 by donaldl@donaldl_crayola_linux_orl

Updated nxt_pos_alloc_incr and nxt_pc_alloc_incr equations to use thread_valid_q
instead of alu_req to validate them.

Change 83323 on 2003/02/10 by hartogs@fl_hartogs

Updated... still in progress.

Change 83322 on 2003/02/10 by hartogs@fl_hartogs

Added unconnected port to modules with new perfmon signals.

Change 83315 on 2003/02/10 by vromaker@vromaker_r400_linux_marlboro

minor updates (a few comment changes, insignificant code change)

Change 83246 on 2003/02/10 by vromaker@vromaker_r400_linux_marlboro

fix for bug 1255 - src c z swizzle set to src c x swizzle for dot2add

Change 83159 on 2003/02/09 by dougd@dougd_r400_linux_marlboro

perfmon signals

Change 83028 on 2003/02/08 by vromaker@vromaker_r400_linux_marlboro

fixed pos and pc alloc terms by using winner_sel instead of winner_ack

Change 83012 on 2003/02/07 by donaldl@donaldl_crayola_linux_orl

Added next parameter cache allocation counter.

Change 83008 on 2003/02/07 by vromaker@vromaker_r400_linux_marlboro

some kill/pred fixes

Change 83001 on 2003/02/07 by dougd@dougd_r400_linux_marlboro

rbi_addr in was shifted down by two bits incorrectly.

Change 82694 on 2003/02/06 by dougd@dougd_r400_linux_marlboro

brought signals up to sq for perfmon

Change 82630 on 2003/02/06 by donaldl@donaldl_crayola_linux_orl

Created exp_buf_empty signal to indicate the export buffers are empty.
This replaces the old SX-SQ export buffer interface signals with the new ones.

Change 82515 on 2003/02/06 by vromaker@vromaker_r400_linux_marlboro

ALU IQ gpr address wrapping; MUL_CONST support; DOT2ADD fix

Change 82428 on 2003/02/06 by donaldl@donaldl_crayola_linux_orl

Created next position allocation counter to determine if all positions have
been allocated for the previous threads.

Change 82207 on 2003/02/05 by vromaker@vromaker_r400_linux_marlboro

tex IQ gpr address wrapping; removal of pop_pending logic

Change 82200 on 2003/02/05 by scamlin@scamlin_crayola_unix_orl

forgot these fuseboxes

Change 82192 on 2003/02/05 by hartogs@fl_hartogs

Changed wire declaration to match associated port declaration.

Change 82190 on 2003/02/05 by hartogs@fl_hartogs

Changed to reflect changes in unit under test.

Change 82186 on 2003/02/05 by hartogs@fl_hartogs

Deleted extra comma at the end of the argument list. THis comma was creating a port mis-match during sim load.

Change 82113 on 2003/02/05 by dougd@dougd_r400_linux_marlboro

internal module signals brought up to sq level for perfmon

Change 81944 on 2003/02/04 by donaldl@donaldl_crayola_linux_orl

Changes for new SX-SQ export buffer availability interface.

Change 81943 on 2003/02/04 by donaldl@donaldl_crayola_linux_orl

Changes for new SX-SQ export buffer availability interface.

Change 81893 on 2003/02/04 by dougd@dougd_r400_linux_marlboro

fix bug in rt wrt logic

Change 81633 on 2003/02/03 by vromaker@vromaker_r400_linux_marlboro

fixed alu phase for ld_pred

Change 81624 on 2003/02/03 by vromaker@vromaker_r400_linux_marlboro

new one-clk ld_pred signal

Change 81559 on 2003/02/03 by vromaker@vromaker_r400_linux_marlboro

misc updates

Change 81558 on 2003/02/03 by dougd@dougd_r400_linux_marlboro

corrected a typo to the last submit

Change 81528 on 2003/02/03 by dougd@dougd_r400_linux_marlboro

added input regs to TP_SQ_thread_id and TP_SQ_stall

Change 81249 on 2003/02/01 by vromaker@vromaker_r400_linux_marlboro

misc updates - need latest merged versions of files...

Change 81227 on 2003/01/31 by vromaker@vromaker_r400_linux_marlboro

inverted kill mask data from SP

Change 81099 on 2003/01/31 by rramsey@rramsey_crayola_linux_orl

change arbitration policy for sq instruction store so each client has
its own phase in the 8 clock cycle, with CP accesses happening
opportunistically

Change 80876 on 2003/01/30 by vromaker@vromaker_r400_linux_marlboro

pred_override update (for waterfall); instr store read requests output from CFS and TIF

Change 80801 on 2003/01/30 by dougd@dougd_r400_linux_marlboro

fixed typos that caused warnings in synopsys

Change 80742 on 2003/01/30 by dougd@dougd_r400_linux_marlboro

removed reset from the register for VGT_send and VGT_event to comply with convention of block input going only to input of one register and no gates

Change 80731 on 2003/01/30 by dougd@dougd_r400_linux_marlboro

added ati_dff_in to ROM_SPx_disable_vtx, TP_SQ_data_rdy, TP_SQ_type

Change 80540 on 2003/01/29 by hartogs@fl_hartogs

Added SC tracker (tbtrk_sc) to the tb_sqsp testbench.

Change 80494 on 2003/01/29 by donaldl@donaldl_crayola_linux_orl

Initial

Change 80177 on 2003/01/28 by mmantor@FL_mmantorLT_r400_win

got basic sc stimulas to work for sq/sp test bench and reset multipass counter during reset

Change 80168 on 2003/01/28 by vromaker@vromaker_r400_linux_marlboro

fix for 48:16 priority mux

Change 80102 on 2003/01/28 by hartogs@fl_hartogs

Added sc_iterator module wrapped specifically for sqsp testbench.

Change 80057 on 2003/01/28 by vromaker@vromaker_r400_linux_marlboro

updates for mova, kill mask, absolute constants

Change 79927 on 2003/01/28 by dougd@dougd_r400_linux_marlboro

fixed bug in vsr_ld_state VSR_LD

Change 79871 on 2003/01/28 by dougd@dougd_r400_linux_marlboro

corrected range on sp_prd_data_q[15:0] from [i] to [i-48] in for loop

Change 79731 on 2003/01/27 by dougd@dougd_r400_linux_marlboro

fixes bug where vsr_ld_state gets stuck in state VSR_LD

Change 79716 on 2003/01/27 by vromaker@vromaker_r400_linux_marlboro

fix for thread arbiter priority encoder; updates to kill mask and predicate loading from SP

Change 79542 on 2003/01/26 by dougd@dougd_r400_linux_marlboro

added inputs i_rbi_rt_rd_req and address logic to support diagnostic read support for Real Time

Change 79540 on 2003/01/26 by dougd@dougd_r400_linux_marlboro

fixed typo that produced latches in synthesis

Change 79498 on 2003/01/25 by vromaker@vromaker_r400_linux_marlboro

updates for kill mask

Change 79443 on 2003/01/25 by hartogs@fl_hartogs

Made two wire declarations match their associated port declarations.

Change 79383 on 2003/01/24 by dougd@dougd_r400_linux_marlboro

changes to SQ_FLOW_CONTROL register

Change 79347 on 2003/01/24 by dougd@dougd_r400_linux_marlboro

fixed bug in vgt interface (end_of_vtx_vector can be valid when indx_valid is not)

Change 78932 on 2003/01/23 by dougd@dougd_r400_linux_marlboro

changes necessary to simulate with rf rams from latest virage compiler

Change 78924 on 2003/01/23 by donaldl@donaldl_crayola_linux_orl

Initial

Change 78850 on 2003/01/23 by scamlin@scamlin_crayola_unix_orl

add rtl versions

Change 78849 on 2003/01/23 by scamlin@scamlin_crayola_unix_orl

add rtl version

Change 78805 on 2003/01/23 by dougd@dougd_r400_linux_marlboro

replaced latest version of this virage rf ram with the one from previous virage compiler because latest version doesn't work

Change 78788 on 2003/01/23 by mmantor@mmantor_crayola_linux_orl

<fixed bugs created by seperation of alloc machine and interp machine to align all data to the sx and sp interfaces and moved thread counter after delay pipe>

Change 78700 on 2003/01/22 by vromaker@vromaker_r400_linux_marlboro

fixes for pred_set/kill; tex_arb_policy added

Change 78428 on 2003/01/22 by scamlin@scamlin_crayola_unix_orl

virage a04 new register files

Change 78427 on 2003/01/22 by scamlin@scamlin_crayola_win

mod for a04

Change 78248 on 2003/01/21 by hartogs@fl_hartogs

Updated for VGT and RBBM stimulus.
Updated for changes in unit under test.

Change 78247 on 2003/01/21 by hartogs@fl_hartogs

Added this port to the instance "uscalar" to file sp/vector/sp_vector.v.
    .oPRED_SET_EXECUTE(), // TODO -- added unconnected output port to avoid warnings during sim load
Apparently the sp_scalar_lut module added an output port (single bit).

Also added these ports to the instance "u_sq_tex_ctl_flow_seq" in sq.v
        .ais_update(),   // listed unconnected port to avoid load warning

.ais_thread_id(), // listed unconnected port to avoid load warning
ais_update is single bit, and ais_thread_id is 6 bits.

Change 78147 on 2003/01/21 by ygiang@ygiang_r400_pv2_marlboro

added: SQ performance counters

Change 77896 on 2003/01/20 by vromaker@vromaker_r400_linux_marlboro

fix for opt. const waterfall (const sel delayed one cycle)

Change 77794 on 2003/01/19 by vromaker@vromaker_r400_linux_marlboro

fix for const waterfall optimization (decode signal was missing a bit); also fix for multiple free_done

Change 77751 on 2003/01/18 by mmantor@mmantor_crayola_linux_orl

<fixed a bug with sc_sample_cntrl by adding an isr register version to mantain during interpolation while the alloc machine moves ahead. Also fixed the context_id in the event fifo for 2clk transfers>

Change 77460 on 2003/01/17 by vromaker@vromaker_r400_linux_marlboro

fix for early push into exec ppb - was pushing conditional executes without testing the condition...

Change 77449 on 2003/01/17 by mmantor@mmantor_crayola_linux_orl

<fixed a buf with ij read address generation when using centers and centroids, sent free_buff a clock earlier so they (cneters and centriods can run at rate continously and added flow control to the alloc sm for max_pass_rd_cnt independant of pb_rts >

Change 77428 on 2003/01/17 by vromaker@vromaker_r400_linux_marlboro

fix for Exec SM state EX_NEXT_CFI - if ppb_op is not ALLOC, assumes it's some type of EXEC and jumps to EXEC

Change 77426 on 2003/01/17 by vromaker@vromaker_r400_linux_marlboro

another fix for tmp_max_rd_pass_cnt (had to be delared as 2 bits before the PPB)

Change 77421 on 2003/01/17 by dougd@dougd_r400_linux_marlboro

added ports to sq_rbbm_interface and wires to support rbbm access for the new sq_perfmon module

Change 77419 on 2003/01/17 by vromaker@vromaker_r400_linux_marlboro

fix for max_rd_pass count

Change 77199 on 2003/01/16 by vromaker@vromaker_r400_linux_marlboro

update for mova interface

Change 77182 on 2003/01/16 by vromaker@vromaker_r400_linux_marlboro

Change 77173 on 2003/01/16 by vromaker@vromaker_r400_linux_marlboro

Change 77133 on 2003/01/16 by scamlin@scamlin_crayola_win

.

Change 77107 on 2003/01/16 by mmantor@mmantor_crayola_linux_orl

<Fixed performance bug preventing at rate interpolation with two interpolants>

Change 76992 on 2003/01/15 by fhsien@fhsien_r400_linux_marlboro

Change MES_ALL parameter for more virage memory

Change 76991 on 2003/01/15 by dougd@dougd_r400_linux_marlboro

fixed typo in last submit of this file

Change 76949 on 2003/01/15 by askende@askende_r400_linux_marlboro

releasing top IO changes related to the new Export Status Control interface between SQ and SX

Change 76829 on 2003/01/15 by dougd@dougd_r400_linux_marlboro

added SX_SP_exp_buf_avail, SX_SQ_exp_count_rdy to support clock gating

Change 76813 on 2003/01/15 by vromaker@vromaker_r400_linux_marlboro

CFS now looks at ais_update to clear alu_instr_pending; also const waterfall fixes

Change 76810 on 2003/01/15 by fhsien@fhsien_r400_linux_marlboro

Change MES_ALL parameter to OFF

Change 76802 on 2003/01/15 by scamlin@fl_regress_p4j_crayola_win

.

Change 76791 on 2003/01/15 by dougd@dougd_r400_linux_marlboro

some of the TP,SP,SX clock gating logic added

Change 76773 on 2003/01/15 by dougd@dougd_r400_linux_marlboro

delete obsolete files

Change 76676 on 2003/01/14 by scamlin@scamlin_crayola_win

update hs ports

Change 76588 on 2003/01/14 by scamlin@scamlin_crayola_win

virage star simulation

Change 76327 on 2003/01/13 by vromaker@vromaker_r400_linux_marlboro

Change 76236 on 2003/01/13 by dougd@dougd_r400_linux_marlboro

fixed bug that loaded v_gpr_addr and v_gpr_base too early

Change 76086 on 2003/01/12 by dougd@dougd_r400_linux_marlboro

adding missing signal to sensitivity list

Change 76016 on 2003/01/11 by vromaker@vromaker_r400_linux_marlboro

fixes for constant waterfalling

Change 75969 on 2003/01/10 by hartogs@fl_hartogs

Added unconnected ports in instantiations to avoid warning message while loading into the simulator.

Change 75887 on 2003/01/10 by dougd@dougd_r400_linux_marlboro

routed gated clocks from sq_rbbm_interface up to sq.v and then back to get around a problem with Power Compiler

Change 75798 on 2003/01/10 by dougd@dougd_r400_linux_marlboro

fixed bug with fifo control when the SP GPRs are full

Change 75472 on 2003/01/09 by dougd@dougd_r400_linux_marlboro

changed "input [0:0] sclk_global" to "input sclk_global" to prevent problems with synopsys

Change 75456 on 2003/01/09 by vromaker@vromaker_r400_linux_marlboro

re-release of AIQ, AIS, and CFS; mova dependency stall added

Change 75419 on 2003/01/08 by hartogs@fl_hartogs

First submission -- no trackers / no injector / no synchro-nothing. Just an SQ and four SP's

Change 75418 on 2003/01/08 by hartogs@fl_hartogs

Fixed not-smart compiler error that shows up in Modelsim (but presumably not in VCS).

Change 75417 on 2003/01/08 by vromaker@vromaker_r400_linux_marlboro

testing

Change 75416 on 2003/01/08 by vromaker@vromaker_r400_linux_marlboro

testing

Change 75398 on 2003/01/08 by vromaker@vromaker_r400_linux_marlboro

what the heck again

Change 75397 on 2003/01/08 by vromaker@vromaker_r400_linux_marlboro

what the heck is going on with sq_alu_instr_seq.v

Change 75344 on 2003/01/08 by dougd@dougd_r400_linux_marlboro

edit comments

Change 75343 on 2003/01/08 by dougd@dougd_r400_linux_marlboro

changes for vgt-sq-sp vertex vector loading performance improvement

Change 75340 on 2003/01/08 by vromaker@vromaker_r400_linux_marlboro

fix for NOP in CFS; reduced triangle size in sq_tests

Change 75023 on 2003/01/07 by dougd@dougd_r400_linux_marlboro

fixed bug in d_rtr when data_cnt == 2

Change 74776 on 2003/01/06 by vromaker@vromaker_r400_linux_marlboro

fix for bug1035: tgt instr fetch updated to pass last_in_group correctly

Change 74489 on 2003/01/04 by dougd@dougd_r400_linux_marlboro

"interp_cnt_q =" was changed to "interp_cnt_q <="

Change 74435 on 2003/01/03 by dougd@dougd_r400_linux_marlboro

removed unused signal from port lists

Change 74403 on 2003/01/03 by vromaker@vromaker_r400_linux_marlboro

fix for bug 1011 (last_in_group/last_in_shader issues with multiple consecutive EXEC's, and multiple instr's in the last EXEC)

Change 74242 on 2003/01/02 by vromaker@vromaker_r400_linux_marlboro

added last optimiztion enable to exec ppb

Change 74199 on 2003/01/02 by vromaker@vromaker_r400_linux_marlboro

more fixes for milestone_tri (gpr_base was off for both gpr read and write)

Change 74166 on 2003/01/02 by vromaker@vromaker_r400_linux_marlboro

fixes for milestone_tri tests

Change 74008 on 2003/01/01 by vromaker@vromaker_r400_linux_marlboro

split the pixel input state machine in two

Change 73136 on 2002/12/23 by dougd@dougd_r400_linux_marlboro

added output o_sq_soft_srst to sq_rbbm_interface

Change 72904 on 2002/12/20 by vromaker@vromaker_r400_linux_marlboro

fix for pix_ctl (buff_swap used instead of free_buff in ptr_buff)

Change 72839 on 2002/12/20 by vromaker@vromaker_r400_linux_marlboro

more pix_ctl (pc) and vtx_ctl (vc) stuff; added some _q's to registered signal names; made performance chages to pix_ctl (pism)

Change 72723 on 2002/12/20 by vromaker@vromaker_r400_linux_marlboro

more updates for pism name change to pix_ctl (PC)

Change 72720 on 2002/12/20 by vromaker@vromaker_r400_linux_marlboro

input arb fix for pix req while pix busy; renamed pism to pix_ctl and vism to vtx_ctl and make new directories pc and vc

Change 72632 on 2002/12/19 by dougd@dougd_r400_linux_marlboro

fixed bug in loading texture constants when there are stalls in the data stream from the rbbm

Change 72217 on 2002/12/18 by vromaker@vromaker_r400_linux_marlboro

fix for CFC read from ALU CFS 1

Change 71817 on 2002/12/17 by vromaker@vromaker_r400_linux_marlboro

Change 71495 on 2002/12/16 by vromaker@vromaker_r400_linux_marlboro

Change 71347 on 2002/12/16 by dougd@dougd_r400_linux_marlboro

Changes for synthesis: added a register stage to sq_const_map_cntl to improve timing; changed direction of TST_SQ_rf_star_fb_out_fusebox in sq.v to output because fusebox is internal to sq

Change 71059 on 2002/12/13 by vromaker@vromaker_r400_linux_marlboro

fix for SC packer optimization (quad ctl interface to SQ)

Change 70462 on 2002/12/11 by dougd@dougd_r400_linux_marlboro

made integer variable names unique for each process block

Change 70353 on 2002/12/11 by dougd@dougd_r400_linux_marlboro

default clause in case statement was 1st in list of cases - it must be last (or synopsys chokes)

Change 70341 on 2002/12/11 by vromaker@vromaker_r400_linux_marlboro

fix for lod_correct bit width change

Change 70330 on 2002/12/11 by dougd@dougd_r400_linux_marlboro

added 1 bit to the width of isr_loop_index_q to make it match it's connections

Change 70179 on 2002/12/10 by vromaker@vromaker_r400_linux_marlboro

update

Change 70178 on 2002/12/10 by vromaker@vromaker_r400_linux_marlboro

fix for consts; top level changes

Change 69908 on 2002/12/10 by dougd@dougd_r400_linux_marlboro

brought SQ_hs_<signals> to gc level and tied to logic low there

Change 69710 on 2002/12/09 by vromaker@vromaker_r400_linux_marlboro

more const waterfall fixes

Change 69670 on 2002/12/09 by vromaker@vromaker_r400_linux_marlboro

misc fixes for vcs compile

Change 69656 on 2002/12/09 by dougd@dougd_r400_linux_marlboro

add hs wrapper files

Change 69651 on 2002/12/09 by vromaker@vromaker_r400_linux_marlboro

constant waterfalling code added to AIQ, AIS, and AIO

Change 69125 on 2002/12/06 by dougd@dougd_r400_linux_marlboro

added hs memories and processors to perforce. added gc level connections to sq.v. ifdef in sq.v to enable hs behavorial memories for simulation.

Change 68379 on 2002/12/04 by dougd@dougd_r400_linux_marlboro

instantiated STAR hs processor, fuse_box and wiring

Change 68158 on 2002/12/03 by vromaker@vromaker_r400_linux_marlboro

fix for event_id to CP (4 to 5 bits); vism fix for continued & end of vector;
addition of ij buffer optimization for centers only/centroids only

Change 67902 on 2002/12/02 by vromaker@vromaker_r400_linux_marlboro

control flow updates; thread buff/status register clean-up

Change 67336 on 2002/11/27 by dougd@dougd_r400_linux_marlboro

fixed typo

Change 67331 on 2002/11/27 by dougd@dougd_r400_linux_marlboro

type in port name of unused input

Change 67098 on 2002/11/26 by vromaker@vromaker_r400_linux_marlboro

ctl flow update

Change 67077 on 2002/11/26 by vromaker@vromaker_r400_linux_marlboro

fixes for milestone_event bugs; functional changes to ctl flow

Change 66948 on 2002/11/26 by dougd@dougd_r400_linux_marlboro

initial submit of sq rf proc and fusebox

Change 66939 on 2002/11/26 by dougd@dougd_r400_linux_marlboro

STAR RF memories for synthesis and simulation: includes the rf proc and fuse_box in sq.v and all of the necessary wiring

Change 66739 on 2002/11/25 by vromaker@vromaker_r400_linux_marlboro

fix for sq_busy

Change 66458 on 2002/11/23 by vromaker@vromaker_r400_linux_marlboro

fix for thread buffer read address wrapping

Change 66229 on 2002/11/22 by dougd@dougd_r400_linux_marlboro

added missing connections between sq_rbbm_interface and the constant store modules

Change 66186 on 2002/11/22 by vromaker@vromaker_r400_linux_marlboro

fix for predicate0_q

Change 66177 on 2002/11/22 by dougd@dougd_r400_linux_marlboro

fixed synopsys warning of cip_q[0] not being in the event list: changed it to cip_q

Change 66043 on 2002/11/21 by vromaker@vromaker_r400_linux_marlboro

fix for bug 818 (thread buffer head and tail ptr wrap problem)

Change 66017 on 2002/11/21 by dougd@dougd_r400_linux_marlboro

add the new Virage STAR rf memories (.v and .ctmc)

Change 65839 on 2002/11/21 by dougd@dougd_r400_linux_marlboro

fix bugs for synthesis. replaced dum_mem_p2 on Virage rf with Virage verilog models for simulation

Change 65700 on 2002/11/20 by vromaker@vromaker_r400_linux_marlboro

fix for bug 799 - a typo in the generation of pvps_disable was causing the problem

Change 65386 on 2002/11/19 by dougd@dougd_r400_linux_marlboro

fix minor coding error introduced in last version

Change 65321 on 2002/11/19 by dougd@dougd_r400_linux_marlboro

fixed a bug in new clock gating logic and changed the clock to the sq_phase_gen module from sclk_sq to sclk_const_mem to generate is_phase when the instruction_store is being loaded from the RBBM and sclk_sq has not yet been turned on. sq_vtx_thread had a port size mismatch on the read addr input

Change 65065 on 2002/11/18 by dougd@dougd_r400_linux_marlboro

added scan test bus. divided sq modules into 9 groups and generated a reset in the rbbm_interface for each group

Change 65064 on 2002/11/18 by dougd@dougd_r400_linux_marlboro

changed default value of parameters to prevent errors in synthesis

Change 65063 on 2002/11/18 by dougd@dougd_r400_linux_marlboro

correct coding errors in the instantiations of the virage memories

Change 64944 on 2002/11/18 by vromaker@vromaker_r400_linux_marlboro

fix for simple_loop

Change 64899 on 2002/11/18 by vromaker@vromaker_r400_linux_marlboro

misc bug fixes

Change 64341 on 2002/11/15 by dougd@dougd_r400_linux_marlboro

fixed bug in clk gater enable signal

Change 64304 on 2002/11/15 by dougd@dougd_r400_linux_marlboro

the ROM_SPx_disable_vtx inputs were not connected to sq_vism.v

Change 64138 on 2002/11/14 by vromaker@vromaker_r400_linux_marlboro

dot2add srcC.z swizzle = x

Change 64003 on 2002/11/14 by dougd@dougd_r400_linux_marlboro

allow the mapping tables to be copied only on the 1st draw command after a gfx_copy_state

Change 63931 on 2002/11/14 by dougd@dougd_r400_linux_marlboro

fixed bug in sq_rd_addr remapping

Change 63882 on 2002/11/13 by dougd@dougd_r400_linux_marlboro

fixed connection errors introduced by merge of files modified to add new Virage STAR memories

Change 63879 on 2002/11/13 by desiree@desiree_r400_sun_marlboro

added Virage STAR memories and interconnections to them

Change 63878 on 2002/11/13 by desiree@desiree_r400_sun_marlboro

added connections for Virage STAR memory in vism_skid_buf

Change 63877 on 2002/11/13 by desiree@desiree_r400_sun_marlboro

added Virage STAR memory

Change 63876 on 2002/11/13 by desiree@desiree_r400_sun_marlboro

added connections to Virage STAR memory in rbbm_skid_buf

Change 63875 on 2002/11/13 by desiree@desiree_r400_sun_marlboro

added Virage STAR memory

Change 63874 on 2002/11/13 by desiree@desiree_r400_sun_marlboro

added Virage STAR memory and interconnect

Change 63873 on 2002/11/13 by desiree@desiree_r400_sun_marlboro

added Virage STAR memory

Change 63870 on 2002/11/13 by desiree@desiree_r400_sun_marlboro

added Virage STAR memories and interconnect

Change 63869 on 2002/11/13 by desiree@desiree_r400_sun_marlboro

added Virage STAR memories and interconnect

Change 63865 on 2002/11/13 by desiree@desiree_r400_sun_marlboro

added STAR Virage memories

Change 63828 on 2002/11/13 by vromaker@vromaker_r400_linux_marlboro

fix for tri32_pix4 (pism was asserting RTR to ptr_buff incorrectly)

Change 63824 on 2002/11/13 by dougd@dougd_r400_linux_marlboro

added bad SP disable logic

Change 63821 on 2002/11/13 by dougd@dougd_r400_linux_marlboro

added rbbm diag read support and fixed RT operation

Change 63820 on 2002/11/13 by dougd@dougd_r400_linux_marlboro

numerous changes to add rbbm read diagnostic feature. Also: corrected the map_copy_context in the const_map_cntl; made all non-constant store rbbm accesses decode before going thru skid buffer.

Change 63494 on 2002/11/12 by vromaker@vromaker_r400_linux_marlboro

removed forcing of PS on _PREV opcodes

Change 63373 on 2002/11/12 by vromaker@vromaker_r400_linux_marlboro

another fix for pspv_wr_en - alu_phase swapped for loading of pipeline registers

Change 63245 on 2002/11/11 by vromaker@vromaker_r400_linux_marlboro

delayed mask and predicate input to pspv_gpr_we by two cycles instead of delaying the alu_phase mux select two cycles

Change 62913 on 2002/11/08 by vromaker@vromaker_r400_linux_marlboro

fix for valid bits (they were swapped in ais_output)

Change 62866 on 2002/11/08 by vromaker@vromaker_r400_linux_marlboro

fixed pspv_gpr write enables

Change 62309 on 2002/11/07 by vromaker@vromaker_r400_linux_marlboro

input arbitration fix - added pipelined pix_wr_busy to PISM

Change 61975 on 2002/11/06 by vromaker@vromaker_r400_linux_marlboro

updates for new gpr write enables

Change 61827 on 2002/11/05 by dougd@dougd_r400_linux_marlboro

make o_sp_vsr_read assert 1 tick earlier to correct a problem with changes in the SQ_SP interface

Change 61651 on 2002/11/05 by vromaker@vromaker_r400_linux_marlboro

more predicate updates including pred_override, and delayed thread buffer
write back

Change 61285 on 2002/11/03 by vromaker@vromaker_r400_linux_marlboro

had to move vtx input grant out one cycle to get the VSR grp writes
lined up with the correct phase

Change 60956 on 2002/11/01 by vromaker@vromaker_r400_linux_marlboro

- removed special case for scalar PREV opcodes
- removed old gpr write enable from SQ_SP interface

Change 60919 on 2002/11/01 by vromaker@vromaker_r400_linux_marlboro

predicate and SP write enable updates

Change 60678 on 2002/10/31 by dougd@dougd_r400_linux_marlboro

remove wire declarations that use `defines for real time paramater registers

Change 60544 on 2002/10/30 by dougd@dougd_r400_linux_marlboro

remove _rt_param* regs to coincide with new emulator

Change 60095 on 2002/10/29 by vromaker@vromaker_r400_linux_marlboro

- added special case for scalar _PREV opcodes
- started predicate updates

Change 59942 on 2002/10/29 by dougd@dougd_r400_linux_marlboro

added "`ifdef" for virage memories

Change 59546 on 2002/10/26 by dougd@dougd_r400_linux_marlboro

added clock gaters and flops for resets

Change 59428 on 2002/10/25 by vromaker@vromaker_r400_linux_marlboro

updates

Change 59279 on 2002/10/25 by vromaker@vromaker_r400_linux_marlboro

minor updates

Change 59186 on 2002/10/24 by dougd@dougd_r400_linux_marlboro

many changes to support RBBM diagnostic read access to the constant memories in aluconst and texconst

Change 59171 on 2002/10/24 by vromaker@vromaker_r400_linux_marlboro

delayed const to SP by one cycle

Change 59080 on 2002/10/24 by vromaker@vromaker_r400_linux_marlboro

- big update for pv/ps detection (it was moved from in front of the
  AIQ to after the AIQ)
- constants fixed to properly add base and index (no waterfalling yet)
- constants fixed to properly map src A,B,C onto c0 and c1
- ALU instr queue uses new reg storage (also, controller was renamed)

Change 58116 on 2002/10/18 by vromaker@vromaker_r400_linux_marlboro

new Instr Queue files

Change 57672 on 2002/10/17 by vromaker@vromaker_r400_linux_marlboro

minor fixes related to CFS state mem expansion

Change 57639 on 2002/10/17 by vromaker@vromaker_r400_linux_marlboro

- more control flow updates
- output rotated requests from vtx thread buffer

Change 57520 on 2002/10/16 by dougd@dougd_r400_linux_marlboro

add 1 bit to size of vsim_event_id

Change 57405 on 2002/10/16 by dougd@dougd_r400_linux_marlboro

fix bug introduced in the previous version that caused o_new_context_ld to go to "X" just after reset

Change 57379 on 2002/10/16 by dougd@dougd_r400_linux_marlboro

fixed bug in decoding GFX_COPY_STATE

Change 57210 on 2002/10/15 by vromaker@vromaker_r400_linux_marlboro

CFS: added control flow instructions

Change 56923 on 2002/10/14 by dougd@dougd_r400_linux_marlboro

added barrel shifter and arbitration winner translation logic needed to remove shifting function of sq_status_reg

Change 56824 on 2002/10/13 by dougd@dougd_r400_linux_marlboro

increase size of pix thread buffer from 16 to 48: fixed bugs in the arbitration winner address translation

Change 56409 on 2002/10/10 by vromaker@vromaker_r400_linux_marlboro

update to cond exec

Change 56155 on 2002/10/09 by vromaker@vromaker_r400_linux_marlboro

updates for conditional execution

Change 56142 on 2002/10/09 by vromaker@vromaker_r400_linux_marlboro

- fix for tp_done/ais_pop collision
- added conditional execution to CFS

Change 55772 on 2002/10/08 by dougd@dougd_r400_linux_marlboro

changed sq_cfc to have three sets of read ports controlled by i_is_sub_phase

Change 55732 on 2002/10/07 by dougd@dougd_r400_linux_marlboro

modified port list of instantiations of sq_status_register to match latest version of that module.

Change 55670 on 2002/10/07 by dougd@dougd_r400_linux_marlboro

increased number of status registers from 16 to 48

Change 55666 on 2002/10/07 by dougd@dougd_r400_linux_marlboro

changed output name of alu(tex)_req_q to alu(tex)_req_out_q on sq_pix_thread_buff

Change 55620 on 2002/10/07 by vromaker@vromaker_r400_linux_marlboro

- fixes to param cache write ptr, and param cache free counter

Change 55389 on 2002/10/04 by vromaker@vromaker_r400_linux_marlboro

param cache write and dealloc update

Change 55202 on 2002/10/03 by dougd@dougd_r400_linux_marlboro

fixes bug reported in Bugzilla: Bug 405 where wrong data was written to sq_instruction_store memory due to interruption or pause of write data on rbbm bus

Change 55025 on 2002/10/03 by dougd@dougd_r400_linux_marlboro

add FIFO_NAME to skid buffer ati_fifo_cntl in rbbm and vism

Change 55017 on 2002/10/03 by dougd@dougd_r400_linux_marlboro

added "FIFO_NAME" to skid buffer ati_fifo_cntl in rbbm and vism

Change 54932 on 2002/10/02 by vromaker@vromaker_r400_linux_marlboro

pism updated for case when param_gen and gen_index detected

Change 54925 on 2002/10/02 by dougd@dougd_r400_linux_marlboro

added the following rbbm register outputs from rbbm_interface: context_misc_sc_output_screen_xy_set, context_misc_sc_sample_cntl_set, gen_index_pix_set and gen_index_vtx_set

Change 54872 on 2002/10/02 by vromaker@vromaker_r400_linux_marlboro

- double buffer for dealloc space added to exit SM so it can handle
  a second dealloc request while the first one is still in progress

Change 54688 on 2002/10/01 by vromaker@vromaker_r400_linux_marlboro

event_vld was collideing with cfs_update : fixed

Change 54647 on 2002/10/01 by vromaker@vromaker_r400_linux_marlboro

- exp_cnt_q (export count) added to sq_export_alloc
- pulse_sx update
- pc_we fixed (export_pc fixed - added thread_type into logic)

Change 54498 on 2002/10/01 by dougd@dougd_r400_linux_marlboro

changed the REN input on the single port use of dum_mem_p2 from 1'b1 to ~wen to prevent the warning messages displayed during simulation

Change 54445 on 2002/09/30 by dougd@dougd_r400_linux_marlboro

changed bit fields on i_addr_in to match rbbm_addr to make a direct match of bit fields (just to make it easier to understand)

Change 54212 on 2002/09/28 by dougd@dougd_r400_linux_marlboro

corrected bit widths on some signals that caused errors in synthesis

Change 54202 on 2002/09/28 by dougd@dougd_r400_linux_marlboro

increased size of vtx_alloc_space because vs_num_reg is 1 less than the actual value we want to req

Change 54201 on 2002/09/28 by dougd@dougd_r400_linux_marlboro

corrected the `defines in the parameter list of the instantiation of sq_thread_buff_cntl from those for "vtx" to those for "pix"

Change 54166 on 2002/09/27 by vromaker@vromaker_r400_linux_marlboro

- SC_SQ interface updates
- also connected VGT_SQ_event to sq_vism

Change 53873 on 2002/09/26 by dougd@dougd_r400_linux_marlboro

defined wires for vss and vdd for virage memories

Change 53800 on 2002/09/26 by vromaker@vromaker_r400_linux_marlboro

- fixes for events flowing thru SQ
- cleared up issues with making individial vtx and pix thread buffers (and shared thread_buff_cntl)
- fixed PV,PS bugs

Change 53737 on 2002/09/25 by dougd@dougd_r400_linux_marlboro

reduced size of cfc constant store

Change 53736 on 2002/09/25 by dougd@dougd_r400_linux_marlboro

added 2 bits to width of vism skid buffer

Change 53735 on 2002/09/25 by dougd@dougd_r400_linux_marlboro

changed sizes of virage rams for mapping tables

Change 53730 on 2002/09/25 by dougd@dougd_r400_linux_marlboro

set all generate flags to "true"

Change 53434 on 2002/09/24 by vromaker@vromaker_r400_linux_marlboro

a few port mismatch fixes

Change 53376 on 2002/09/24 by dougd@dougd_r400_linux_marlboro

removed redundant declaration that caused synopsys compile error

Change 53375 on 2002/09/24 by vromaker@vromaker_r400_linux_marlboro

- fixes for moving event thru the SQ
- fixes for dealloc, and state_diff in thread buffers

Change 53204 on 2002/09/23 by dougd@dougd_r400_linux_marlboro

corrected mix of assigments: next_old_context <= old_context_q;   // synopsys doesn't like "<=" mixed with "="

Change 53136 on 2002/09/23 by dougd@dougd_r400_linux_marlboro

remove "wire [7:0] temp6 = i_addr_in/6;" which cause synthesis error

Change 53039 on 2002/09/23 by dougd@dougd_r400_linux_marlboro

new modules to increase size of pixel thread buffer

Change 52738 on 2002/09/20 by vromaker@vromaker_r400_linux_marlboro

event fifo to pism/ptb fixes

Change 52350 on 2002/09/18 by vromaker@vromaker_r400_linux_marlboro

ptr buff fix to work correctly with split 2-cycle transfers

Change 52270 on 2002/09/18 by dougd@dougd_r400_linux_marlboro

corrected width of constant assigned to alloc_size_q from 2 to 4

Change 52212 on 2002/09/18 by dougd@dougd_r400_linux_marlboro

modified fix to sq_valid_2_q that was entered in the previous version

Change 52164 on 2002/09/17 by dougd@dougd_r400_linux_marlboro

added more to fix for sq_qual_2_q of previous version

Change 52160 on 2002/09/17 by dougd@dougd_r400_linux_marlboro

make sc_valid_2_q stay asserted until there is a SC_SQ_valid

Change 51789 on 2002/09/16 by dougd@dougd_r400_linux_marlboro

fixed bug with SQ_RBB_rs outputting x's; fixed bug with code added to support rbbm diagnostic read of constant store memories

Change 51740 on 2002/09/16 by tien@tien_r400_devel_marlboro

Interface change for post-June 15th inst/const

Change 51675 on 2002/09/15 by dougd@dougd_r400_linux_marlboro

fixed bug in rbbm diagnostic read interface

Change 51559 on 2002/09/13 by dougd@dougd_r400_linux_marlboro

connect acs_rd_req to sq_aluconst_top (that input was floating)

Change 51526 on 2002/09/13 by vromaker@vromaker_r400_linux_marlboro

- gpr dealloc connected
- static gpr allocation added (but not enabled)
- ppb btwn pism and ptb added

Change 51368 on 2002/09/13 by dougd@dougd_r400_linux_marlboro

added some of the port connections necessary to support RBBM reading of the constant store memories

Change 50967 on 2002/09/12 by dougd@dougd_r400_linux_marlboro

changed port name "i_context_switch" to "i_map_copy_start" for aluconst and texconst; changed "i_state_change_flag" to "i_read_base_ld" and added ports based on state change based on gfx_copy_state to sq_cfc

Change 50916 on 2002/09/11 by dougd@dougd_r400_linux_marlboro

connect context_switch based on gfx_copy_state in rbi; connect loading mechanism for eo_rt start addresses for aluconst and texconst

Change 50806 on 2002/09/11 by vromaker@vromaker_r400_linux_marlboro

fixes for bug326 and 329 - tests still fail, but for different reasons

Change 50723 on 2002/09/11 by dougd@dougd_r400_linux_marlboro

added support for real time mode

Change 50564 on 2002/09/10 by vromaker@vromaker_r400_linux_marlboro

update

Change 50503 on 2002/09/10 by vromaker@vromaker_r400_linux_marlboro

another PV/PS phase swap bug fix

Change 50458 on 2002/09/10 by dougd@dougd_r400_linux_marlboro

these files moved to directory where they are used

Change 50294 on 2002/09/09 by vromaker@vromaker_r400_linux_marlboro

update to write enables due to PV, PS cycle swap

Change 50193 on 2002/09/09 by vromaker@vromaker_r400_linux_marlboro

updated kill_mask out to SX

Change 50165 on 2002/09/09 by vromaker@vromaker_r400_linux_marlboro

fix for pc write one cycle early

Change 50034 on 2002/09/06 by dougd@dougd_r400_linux_marlboro

initial submission of skid buf ram for sq_rbbm_interface

Change 49970 on 2002/09/06 by vromaker@vromaker_r400_linux_marlboro

minor udpates

Change 49848 on 2002/09/05 by vromaker@vromaker_r400_linux_marlboro

- added predicate, kill mask, pv/ps detection
- swapped PV and PS write gpr phase

Change 49671 on 2002/09/05 by dougd@dougd_r400_sun_marlboro

changed sizes of new_map_ram, map_ram and freelist to cover full size of texconst_mem

Change 49291 on 2002/09/03 by dougd@dougd_r400_linux_marlboro

removed context_misc_screen_xy_in_gpr0_set from sq.v and sq_rbbm_interface.v. added address decoding for real time constants in sq_rbbm_interface.v

Change 49226 on 2002/09/02 by dougd@dougd_r400_sun_marlboro

initial checkin

Change 49220 on 2002/09/02 by dougd@dougd_r400_linux_marlboro

brought in 3 more bits of rbi_addr and divide it by 6 to get the correct texture constant address because the 6 Dwords in each constant are no longer packed on boundaries of 8 Dwords but on boundaries of 6 Dwords.

Change 49065 on 2002/08/30 by dougd@dougd_r400_linux_marlboro

add "vs_num_reg + 1" logic and increase port size by 1 bit

Change 49059 on 2002/08/30 by vromaker@vromaker_r400_linux_marlboro

fixed a few typos for the new SP instruction decode

Change 48976 on 2002/08/30 by dougd@dougd_r400_linux_marlboro

make o_v_ld_cntl_pkt deassert the next clk after receiving i_vtb_rtr

Change 48974 on 2002/08/30 by vromaker@vromaker_r400_linux_marlboro

- needed to drive acfs_reading one cycle earlier for ACFS IS read
- updated/added new SQ_SP instruction interface

Change 48932 on 2002/08/29 by dougd@dougd_r400_linux_marlboro

replaced address constant with value defined in sq_reg.v

Change 48844 on 2002/08/29 by dougd@dougd_r400_linux_marlboro

added support for gen_index (auto-count), vgt events and fixed some bugs

Change 48558 on 2002/08/28 by vromaker@vromaker_r400_linux_marlboro

- fix for out-of-order thread processing: the 2 alu ctl flow sequencers
  now share one instr store read slot instead of alternating between two
  different slots (which allowed one to get ahead opf the other)
- thread counts from VISM and PISM to ais_output added at SQ level

Change 48384 on 2002/08/27 by vromaker@vromaker_r400_linux_marlboro

- updates for ptr_buff/pism to align quad mask correctly
- additions for thread_count

Change 48164 on 2002/08/26 by vromaker@vromaker_r400_linux_marlboro

- fixes for individual macc write enables

- added the prev_pos_alloc inputs to the status regs (and logic to
  generate them in the tread buffer)

Change 47553 on 2002/08/22 by vromaker@vromaker_r400_linux_marlboro

- ptr buff changed for out-of-order quads
- pism: now add 1 to ps_num_reg to get the number of GPRs to alloc

Change 47160 on 2002/08/20 by vromaker@vromaker_r400_linux_marlboro

connected param_gen_pos to pism

Change 47110 on 2002/08/20 by dougd@dougd_r400_linux_marlboro

added the two rbbm registers that were missed in the last version

Change 47072 on 2002/08/20 by vromaker@vromaker_r400_linux_marlboro

updated param_wrap wires

Change 46976 on 2002/08/20 by dougd@dougd_r400_linux_marlboro

adding the remaining rbbm register outputs to sq_rbbm_interface and wired them up in sq.v

Change 46800 on 2002/08/19 by vromaker@vromaker_r400_linux_marlboro

updated pism connections to local registers

Change 46714 on 2002/08/19 by vromaker@vromaker_r400_linux_marlboro

updated local register inputs to PISM

Change 46643 on 2002/08/16 by dougd@dougd_r400_linux_marlboro

added vgt_event to port list

Change 46642 on 2002/08/16 by dougd@dougd_r400_linux_marlboro

added register outputs from rbbm_interface and vgt_event from sq_vism

Change 46637 on 2002/08/16 by vromaker@vromaker_r400_linux_marlboro

fix for alloc size

Change 46629 on 2002/08/16 by vromaker@vromaker_r400_linux_marlboro

more fixes for alloc size

Change 46574 on 2002/08/16 by vromaker@vromaker_r400_linux_marlboro

fix for SQ_SX_export_id (was connected to wrong signal)

Change 46517 on 2002/08/16 by vromaker@vromaker_r400_linux_marlboro

fixed thread read state machine typo

Change 46382 on 2002/08/15 by vromaker@vromaker_r400_linux_marlboro

fixed pop_thread to be only one cycle

Change 46251 on 2002/08/15 by vromaker@vromaker_r400_linux_marlboro

updates for pop/winner_ack status reg conflict

Change 45784 on 2002/08/13 by dougd@dougd_r400_linux_marlboro

fixed synchronization of writes to RAM by removing input flop on i_texconst_phase to be compatible with same change made in sq_texconst_mem.v sometime ago. Also updated local testbench for sq_texconst block.

Change 45466 on 2002/08/12 by askende@askende_r400_sun_marlboro

checking in with Vic's permission changes related to vsr_vu_valid

Change 45406 on 2002/08/12 by dougd@dougd_r400_linux_marlboro

add change to deassert q_ins_sel when i_ins_rtr is returned.

Change 45291 on 2002/08/09 by dougd@dougd_r400_linux_marlboro

removed "[0:0]" from "input [0:0] clk;" in sq_status_reg.v to prevent synopsys tcl script error during synthesis. Removed divide-by-3 code in sq_instruction_store.v to prevent synthesis error.

Change 45271 on 2002/08/09 by dougd@dougd_r400_linux_marlboro

add i_texconst_rtr to deassert q_tex_sel

Change 45079 on 2002/08/08 by efong@efong_crayola_linux_cvd

removed all the hacked files

Change 44548 on 2002/08/06 by vromaker@vromaker_r400_linux_marlboro

- status register shift connection bug fixed

Change 44376 on 2002/08/06 by dougd@dougd_r400_linux_marlboro

changed default parameter values STATE_WIDTH =64; CFS_STATE_WIDTH = 32; STATUS_WIDTH = 32; to prevent index select errors in synthesis

Change 44356 on 2002/08/06 by dougd@dougd_r400_linux_marlboro

changed default parameter value of 16 to STATUS_WIDTH = 32; to prevent error: slice direction does not match array direction in synthesis

Change 44355 on 2002/08/06 by dougd@dougd_r400_linux_marlboro

changed default parameter values (was 8): STATE_WIDTH = 64; STATUS_WIDTH = 32; so that select index would not be out of bounds and cause synthesis to error

Change 44314 on 2002/08/05 by vromaker@vromaker_r400_linux_marlboro

more delay for free_done

Change 44294 on 2002/08/05 by vromaker@vromaker_r400_linux_marlboro

- 3 cycle delay added for free_done
- port width fixes

Change 44234 on 2002/08/05 by sallen@sallen_r400_lin_marlboro

ferret: finish up backdoor ucode loading, pli changes, etc

Change 44201 on 2002/08/05 by vromaker@vromaker_r400_linux_marlboro

- free_done fix: don't send on param_cache (vtx shdr) done
- sq: added SQ_SP_vsr_vu_valid
- updates to VISM to handle end_of_vector with invalid data

Change 44010 on 2002/08/02 by vromaker@vromaker_r400_linux_marlboro

- multi pixel vector fixes
- VISM fixed for 32 vertex test

Change 43237 on 2002/07/30 by vromaker@vromaker_r400_linux_marlboro

- temp fix to ptr buff to delay free_buff to SC
- comments in thread arb
- re-enabled alu interleaving

Change 42997 on 2002/07/29 by vromaker@vromaker_r400_linux_marlboro

- input arb now grants pix while pix is busy
    - pism skips idle if request is present
- interleaving disabled in sq.v

Change 42996 on 2002/07/29 by vromaker@vromaker_r400_linux_marlboro

- fixed priority encoders (was reversed)

Change 42684 on 2002/07/26 by vromaker@vromaker_r400_linux_marlboro

- reverted valid_bits to go from lsb to msb

Change 42415 on 2002/07/25 by dougd@dougd_r400_linux_marlboro

added ati_rbbm_intf to complete the RBB_rd path

Change 42246 on 2002/07/24 by vromaker@vromaker_r400_linux_marlboro

- fixed thread_id width (caused 2nd pix vector to be same as 1st)

Change 42150 on 2002/07/24 by vromaker@vromaker_r400_linux_marlboro

- fixed ais_acs_rd_addr for synthesis

Change 42144 on 2002/07/24 by vromaker@vromaker_r400_linux_marlboro

- thread_id width fixes

Change 42107 on 2002/07/23 by markf@markf_r400_linux_marlboro

Updated SC->SQ interface

Change 42096 on 2002/07/23 by vromaker@vromaker_r400_linux_marlboro

- forced sq-tp pix_mask to 0xF

Change 42084 on 2002/07/23 by vromaker@vromaker_r400_linux_marlboro

- fixed SQ_SC interface connections

Change 42069 on 2002/07/23 by vromaker@vromaker_r400_linux_marlboro

- reversed order of valid_bits (aka pix_mask)

Change 41959 on 2002/07/23 by vromaker@vromaker_r400_linux_marlboro

- right shift 1 into MSB of valid_bit string (instead of left shift into LSB)

Change 41839 on 2002/07/22 by vromaker@vromaker_r400_linux_marlboro

- new, wider SC interface

```
Change 41838 on 2002/07/22 by vromaker@vromaker_r400_linux_marlboro

        delete

Change 41831 on 2002/07/22 by dougd@dougd_r400_linux_marlboro

        added `include "../misc/sq_defs.v"

Change 41826 on 2002/07/22 by dougd@dougd_r400_linux_marlboro

        changed parameter STATUS_WIDTH value from 4 to 16 to prevent compilation
problems in synthesis

Change 41823 on 2002/07/22 by dougd@dougd_r400_linux_marlboro

        changed order of output declarations to come before their reg declarations so that
synopsys would not declare the outputs as wires

Change 41804 on 2002/07/22 by dougd@dougd_r400_linux_marlboro

        created sq_rbbm_skid_buf with virage mem to replace ati_skid_buff

Change 41796 on 2002/07/22 by vromaker@vromaker_r400_linux_marlboro

        - make the thread_id width consistent at 6 bits (except at the state mem address port)
        - updated the SQ_TP and TP_SQ interface (got rid of SQ_TP_clause_num)

Change 41748 on 2002/07/22 by vromaker@vromaker_r400_linux_marlboro

        fixed state width for sythesis

Change 41592 on 2002/07/19 by vromaker@vromaker_r400_linux_marlboro

        - interleaving is enabled
        - fix for interleaving: cfs_type strap on ALU CFS 1 corrected to 2

Change 41459 on 2002/07/19 by vromaker@vromaker_r400_linux_marlboro

        - more thread_id updates due to new location of thread_id in status register

Change 41453 on 2002/07/19 by vromaker@vromaker_r400_linux_marlboro

        - ppb logic fix
             - fixed updated field position of thread_id w/in status (was causing
               a state_mem read address error since SMRA = winner[status[thread_id]]

Change 41326 on 2002/07/18 by vromaker@vromaker_r400_linux_marlboro

        - corrected exp_type for pix w/o z
```

```
        - fixed cfs_export_id_q to load global_export_id_q only when allocating
        - or'd more signals together in TIF to get a solid busy output

Change 41297 on 2002/07/18 by dougd@dougd_r400_linux_marlboro

        fix typo in previous checkin

Change 41259 on 2002/07/18 by dougd@dougd_r400_linux_marlboro

        intitial checkin of skid buffer used in sq_vism.v

Change 41218 on 2002/07/18 by dougd@dougd_r400_linux_marlboro

        more changes to support synthesis

Change 41217 on 2002/07/18 by vromaker@vromaker_r400_linux_marlboro

        - fixes for sq-sx export

Change 41188 on 2002/07/17 by efong@efong_crayola_linux_cvd

        put in `endif

Change 40943 on 2002/07/16 by dougd@dougd_r400_linux_marlboro

        original submission of virage memory *.ctmc files. The *.v files were modified to
support synthesis.

Change 40937 on 2002/07/16 by vromaker@vromaker_r400_linux_marlboro

        - added alu_instr_pending status bit
        - added new SQ_SX_exp and SQ_SX_free interfaces (free is not functional)

Change 40686 on 2002/07/15 by vromaker@vromaker_r400_linux_marlboro

        - updated decode for exports to be the same as in the AIQ: this
          fixes extraneous GPR writes

Change 40659 on 2002/07/15 by vromaker@vromaker_r400_linux_marlboro

        - added 2nd alu cfs update interface to thread buff
        - state read addr now status_thread_id[winner] as it should have been
        - reg'd cfs_phase in thread buff to match reg'd update data

Change 39972 on 2002/07/12 by vromaker@vromaker_r400_linux_marlboro

        fixes for 2 pixel vectors

Change 39731 on 2002/07/11 by vromaker@vromaker_r400_linux_marlboro
```

```
        fixes for 2 pix vectors

Change 39002 on 2002/07/09 by vromaker@vromaker_r400_linux_marlboro

        misc

Change 38998 on 2002/07/09 by vromaker@vromaker_r400_linux_marlboro

        temp file

Change 38997 on 2002/07/09 by vromaker@vromaker_r400_linux_marlboro

        not sure - checked in due to clean up

Change 36278 on 2002/06/25 by dougd@dougd_r400_linux_marlboro

        added input VGT_SQ_event; changed VGT_SQ_vsisr_double to
VGT_SQ_vsisr_continued

Change 36192 on 2002/06/25 by dougd@dougd_r400_linux_marlboro

        added connections and function to support SQ_RBBM_cntx17_busy &
SQ_RBBM_cntx0_busy, however, at this time both of these signals are the same

Change 36176 on 2002/06/25 by markf@markf_r400_linux_marlboro

        Tied SQ_RBBM_nrtrtr to SQ_RBBM_rtr

Change 35120 on 2002/06/20 by vromaker@vromaker_r400_linux_marlboro

        changes for latest emulator

Change 35005 on 2002/06/19 by vromaker@vromaker_r400_linux_marlboro

        more busy bits

Change 34969 on 2002/06/19 by vromaker@vromaker_r400_linux_marlboro

        fix for CFI fetch (alloc had to update CFI ptr); added a few busy signals

Change 34833 on 2002/06/18 by vromaker@vromaker_r400_linux_marlboro

        fix for wrong thread type

Change 34806 on 2002/06/18 by vromaker@vromaker_r400_linux_marlboro

        took away 4 cycles of delay on pix_gpr_wr{addr, en}
```

```
Change 34778 on 2002/06/18 by vromaker@vromaker_r400_linux_marlboro

        fix for pix shader alu instruction

Change 34632 on 2002/06/17 by dougd@dougd_r400_linux_marlboro

        added a full subtract of the instruction store base address from the rbi_addr before doing
the divide by 3 to get the memory addr

Change 34631 on 2002/06/17 by vromaker@vromaker_r400_linux_marlboro

        hack to delay SC input 16 cycles

Change 34606 on 2002/06/17 by dougd@dougd_r400_linux_marlboro

        commented out the change made in the last version because it needs to be released at the
same time as another change in the sq to work properly

Change 34588 on 2002/06/17 by vromaker@vromaker_r400_linux_marlboro

        added delays for SQ_SP_interp ctl and SQ_SP_gpr_write for interp data

Change 34547 on 2002/06/17 by dougd@dougd_r400_linux_marlboro

        fixed bug in o_vector_valid where it was setting one too many bits.

Change 34539 on 2002/06/17 by vromaker@vromaker_r400_linux_marlboro

        temp hack to param cache write addr and enable to move them out 1 cycle

Change 34347 on 2002/06/15 by vromaker@vromaker_r400_linux_marlboro

        fixes for sending interp ctl to SX/SP

Change 34111 on 2002/06/14 by vromaker@vromaker_r400_linux_marlboro

        got rid of temp hack

Change 34086 on 2002/06/14 by rbell@crayola_misc_linux

        Fixed runsim to return rc no larger than 255.
        Fixes for the full chip build

Change 34083 on 2002/06/14 by vromaker@vromaker_r400_linux_marlboro

        sending correct export address in SP instruction

Change 34062 on 2002/06/14 by dougd@dougd_r400_linux_marlboro
```

replaced the v2k indexed part select implementation with muxes

Change 33977 on 2002/06/13 by vromaker@vromaker_r400_linux_marlboro

changed polarity of exp_pix

Change 33940 on 2002/06/13 by vromaker@vromaker_r400_linux_marlboro

many updates... some v2k removal

Change 33853 on 2002/06/13 by rbell@rbell_crayola_sun_cvd

Had to create more "hacked" files...port mismatches. Must be fixed later

Change 33801 on 2002/06/13 by rbell@rbell_crayola_sun_cvd

Fixes/hacks to get the first chip integration compile to work.

Change 33723 on 2002/06/13 by dougd@dougd_r400_linux_marlboro

added context_valid from aluconst_top to sq_vism to enable/stall loading of control packet from vgt until the alu constant store has been loaded for this state.

Change 33615 on 2002/06/12 by vromaker@vromaker_r400_linux_marlboro

misc updates... alu_req logic updated in sq_status_reg

Change 33554 on 2002/06/12 by vromaker@vromaker_r400_linux_marlboro

moved gpr_rd_en one cycle earlier for srcA

Change 33536 on 2002/06/12 by vromaker@vromaker_r400_linux_marlboro

sending srcA gpr read addr one cycle earlier

Change 33519 on 2002/06/12 by dougd@dougd_r400_linux_marlboro

fix bug in o_context_valid being set correctly

Change 33509 on 2002/06/12 by vromaker@vromaker_r400_linux_marlboro

fixed exporting bit by putting pred_sel bit in correctly

Change 33492 on 2002/06/12 by vromaker@vromaker_r400_linux_marlboro

various updates - instr start asserted to SP

Change 33348 on 2002/06/11 by vromaker@vromaker_r400_linux_marlboro

fixed tex instruction read pointer

Change 33233 on 2002/06/11 by vromaker@vromaker_r400_linux_marlboro

SX exp added; tgt instr cnt from CFS to TIF fixed; alloc stuff added

Change 32898 on 2002/06/10 by vromaker@vromaker_r400_linux_marlboro

fixed sq-sp gpr_rd_en; changed "state" to "context_id" in instr pipes

Change 32795 on 2002/06/07 by vromaker@vromaker_r400_linux_marlboro

more updates

Change 32774 on 2002/06/07 by dougd@dougd_r400_linux_marlboro

fix typo bug in last version

Change 32767 on 2002/06/07 by dougd@dougd_r400_linux_marlboro

added input i_vtb_rtr to complement o_v_ld_cntl_pkt to form handshake

Change 32678 on 2002/06/07 by dougd@dougd_r400_linux_marlboro

fix bug in address decode logic

Change 32472 on 2002/06/06 by vromaker@vromaker_r400_linux_marlboro

thread buff - arb interface updates

Change 32366 on 2002/06/06 by dougd@dougd_r400_linux_marlboro

initial checkin

Change 32295 on 2002/06/06 by vromaker@vromaker_r400_linux_marlboro

commented out fsdbdumpmem

Change 32275 on 2002/06/06 by vromaker@vromaker_r400_linux_marlboro

updated tex instr const_index field to the new format

Change 32269 on 2002/06/06 by dougd@dougd_r400_linux_marlboro

remove input register on i_texconst_phase to sync data xfer to sq

Change 32225 on 2002/06/06 by dougd@dougd_r400_linux_marlboro

fix bug in o_context_switch in sq_rbbm_interface and set map_copy_cntr to 3'd7 at reset in sq_const_map_cntl

Change 32159 on 2002/06/05 by dougd@dougd_r400_linux_marlboro

add decode of gfx_draw_initiator to rbbm_interface to generate context switch to force a map_copy operation in const_map_cntl

Change 32104 on 2002/06/05 by vromaker@vromaker_r400_linux_marlboro

connected SQ_TP_send to internal SQ_TP_vld

Change 31996 on 2002/06/05 by dougd@dougd_r400_linux_marlboro

o_v_grp_addr was being incremented 1 cycle too early. Fixed.

Change 31953 on 2002/06/05 by vromaker@vromaker_r400_linux_marlboro

updated texture pipe output format

Change 31884 on 2002/06/04 by dougd@dougd_r400_linux_marlboro

initial checkin

Change 31883 on 2002/06/04 by dougd@dougd_r400_linux_marlboro

fixed bug

Change 31880 on 2002/06/04 by dougd@dougd_r400_linux_marlboro

changed the timing of the CP write to use the same non-registered input address mux as the reads

Change 31875 on 2002/06/04 by vromaker@vromaker_r400_linux_marlboro

updates

Change 31866 on 2002/06/04 by dougd@dougd_r400_linux_marlboro

added connections to o_inst_base_vtx and o_inst_base_pix

Change 31821 on 2002/06/04 by dougd@dougd_r400_linux_marlboro

fix bug in previous version

Change 31818 on 2002/06/04 by dougd@dougd_r400_linux_marlboro

removed register stage for address into RAM

Change 31805 on 2002/06/04 by dougd@dougd_r400_linux_marlboro

connected o_is_data to read_data

Change 31700 on 2002/06/04 by dougd@dougd_r400_linux_marlboro

changed <= to = in combinatorial blocks to satisfy Leda

Change 31699 on 2002/06/04 by dougd@dougd_r400_linux_marlboro

initial checkin of useful files

Change 31693 on 2002/06/04 by vromaker@vromaker_r400_linux_marlboro

updates

Change 31621 on 2002/06/03 by vromaker@vromaker_r400_linux_marlboro

updates

Change 31586 on 2002/06/03 by vromaker@vromaker_r400_linux_marlboro

updates

Change 31449 on 2002/06/03 by dougd@dougd_r400_linux_marlboro

made temporary fix (marked with FIXME comment) to continue using TP_SQ_clause_num in the port list instead of the newer (replacement) TP_SQ_thread_id which was declared a wire set to "0" to keep gc_test.v working.

Change 31428 on 2002/06/03 by dougd@dougd_r400_linux_marlboro

added tempory wire o_vs_base_set = o_vs_program_base_set;

Change 31427 on 2002/06/03 by dougd@dougd_r400_linux_marlboro

replaced i_cf_addr with i_alu0_cf_addr, i_alu1_cf_addr, i_tex_cf_addr and replaced i_alu_phase with i_is_sub_phase.

Change 31389 on 2002/06/03 by vromaker@vromaker_r400_linux_marlboro

updated

Change 31361 on 2002/06/02 by vromaker@vromaker_r400_linux_marlboro

updates

Change 31279 on 2002/05/31 by vromaker@vromaker_r400_linux_marlboro

updates

Change 31031 on 2002/05/31 by dougd@dougd_r400_linux_marlboro

added functionality for o_vs_first_thread

Change 30987 on 2002/05/30 by dougd@dougd_r400_linux_marlboro

added vs_instr_ptr, vs_resource and vs_first_thread as outputs from sq_vism

Change 30971 on 2002/05/30 by vromaker@vromaker_r400_linux_marlboro

updates

Change 30816 on 2002/05/30 by vromaker@vromaker_r400_linux_marlboro

fixed blocking assignment on SQ_SP_gpr_wr_en

Change 30762 on 2002/05/29 by dougd@dougd_r400_linux_marlboro

fixed various bugs

Change 30562 on 2002/05/29 by vromaker@vromaker_r400_linux_marlboro

fixed input_sel output

Change 30559 on 2002/05/29 by vromaker@vromaker_r400_linux_marlboro

connected the gpr input mux sel

Change 30516 on 2002/05/29 by dougd@dougd_r400_linux_marlboro

added connection to gen_index_set output from sq_rbbm_interface

Change 30462 on 2002/05/28 by dougd@dougd_r400_linux_marlboro

o_v_gpr_we was "X" so hardwired o_v_gpr_we = 1'b1; as a temporary fix.

Change 30458 on 2002/05/28 by vromaker@vromaker_r400_linux_marlboro

updates

Change 30340 on 2002/05/28 by dougd@dougd_r400_linux_marlboro

wired up outputs from new registers to old versions of same outputs. This is tempory until we switch over completely to the new register spec.

Change 30289 on 2002/05/28 by dougd@dougd_r400_linux_marlboro

added output "o_context_switch" to sq_rbbm_interface and connected it to sq_aluconst_top and sq_texconst_top in sq.v
<enter description here>

Change 30286 on 2002/05/28 by vromaker@vromaker_r400_linux_marlboro

removing from tis

Change 30284 on 2002/05/28 by vromaker@vromaker_r400_linux_marlboro

moved file from tis to cfs

Change 30282 on 2002/05/28 by vromaker@vromaker_r400_linux_marlboro

updates...

Change 30159 on 2002/05/27 by vromaker@vromaker_r400_linux_marlboro

updates

Change 30053 on 2002/05/24 by dougd@dougd_r400_linux_marlboro

rbi_acs_rts was wired to both o_aluconst_rts and o_texconst_rts from sq_rbbm_interface: fixed

Change 30048 on 2002/05/24 by vromaker@vromaker_r400_linux_marlboro

checkpoint update

Change 30021 on 2002/05/24 by dougd@dougd_r400_linux_marlboro

extended duration of i_map_copy_active to hold rtr inactive 1 more tick to allow pa to be allocated.

Change 29966 on 2002/05/24 by dougd@dougd_r400_linux_marlboro

changed include file

Change 29948 on 2002/05/24 by dougd@dougd_r400_linux_marlboro

sq_rbbm_interface supports both old and new register `defines and has all the new state registers. sq.v instantiates this sq_rbbm_interface.

Change 29802 on 2002/05/23 by dougd@dougd_r400_linux_marlboro

fixed various bugs

Change 29768 on 2002/05/23 by vromaker@vromaker_r400_linux_marlboro

initial version

Change 29767 on 2002/05/23 by vromaker@vromaker_r400_linux_marlboro

updates

Change 29750 on 2002/05/23 by vromaker@vromaker_r400_linux_marlboro

updates.. now has clk and reset inputs...

Change 29311 on 2002/05/21 by vromaker@vromaker_r400_linux_marlboro

added SQ_CTL_PKT_WIDTH back in

Change 29136 on 2002/05/20 by vromaker@vromaker_r400_linux_marlboro

updates...

Change 28916 on 2002/05/17 by vromaker@vromaker_r400_linux_marlboro

new sq files for clause-less state management : initial, not complete, versions

Change 28866 on 2002/05/17 by dougd@dougd_r400_linux_marlboro

minor logic fixes

Change 28533 on 2002/05/16 by dougd@dougd_r400_linux_marlboro

tempory use to allow compile until new register spec is implemented.

Change 28531 on 2002/05/16 by dougd@dougd_r400_linux_marlboro

added temporary include of ../sq_reg_old.v to allow compile until the new register spec is implemented

Change 27919 on 2002/05/14 by dougd@dougd_r400_linux_marlboro

this is the old register spec which is needed while we are still using rtl based on this spec

Change 27917 on 2002/05/14 by dougd@dougd_r400_sun_marlboro

changed `include "register_addr.v to `include "../sq_register_addr.v to allow compilation of rtl based on old register spec

Change 27837 on 2002/05/14 by dougd@dougd_r400_linux_marlboro

added prefix sq_ to module and file name

Change 27828 on 2002/05/14 by vromaker@vromaker_r400_linux_marlboro

added fifo_regs_ctl to sq/misc

Change 27332 on 2002/05/10 by dougd@dougd_r400_sun_marlboro

added a divide by 3 to the incoming RBI address to generate the correct instruction memory address

Change 27179 on 2002/05/09 by dougd@dougd_r400_sun_marlboro

changed size of outputs o_inst_base_vtx and o_inst_base_pix from 8x because they are not state(context) registers

Change 27099 on 2002/05/08 by dougd@dougd_r400_sun_marlboro

added outputs for the initial set of state registers in sq.v

Change 27093 on 2002/05/08 by dougd@dougd_r400_sun_marlboro

changed the values assigned to i_is_phase

Change 27092 on 2002/05/08 by dougd@dougd_r400_sun_marlboro

added sq_ as prefix to module and file names

Change 27088 on 2002/05/08 by dougd@dougd_r400_sun_marlboro

added sq_ as prefix to module and file

Change 27087 on 2002/05/08 by dougd@dougd_r400_sun_marlboro

added sq_ prefix to module and file names

Change 27050 on 2002/05/08 by vromaker@vromaker_r400_linux_marlboro

updates

Change 26913 on 2002/05/08 by dougd@dougd_r400_sun_marlboro

this module was renamed to sq_instruction_store.v

Change 26907 on 2002/05/08 by dougd@dougd_r400_sun_marlboro

this file was renamed to sq_vism.v

Change 26905 on 2002/05/08 by dougd@dougd_r400_sun_marlboro

changed some IO names

Change 26903 on 2002/05/08 by dougd@dougd_r400_sun_marlboro

changed module name from vism to sq_vism. changed some IO names.

Change 26852 on 2002/05/07 by dougd@dougd_r400_sun_marlboro

renamed module from is.v to sq_instruction_store.v

Change 26785 on 2002/05/07 by dougd@dougd_r400_sun_marlboro

initial version is incomplete and in development.

Change 26731 on 2002/05/07 by vromaker@vromaker_r400_linux_marlboro

delete

Change 26729 on 2002/05/07 by vromaker@vromaker_r400_linux_marlboro

delete

Change 26726 on 2002/05/07 by vromaker@vromaker_r400_sun_marlboro

submitting all...

Change 26717 on 2002/05/07 by vromaker@vromaker_r400_sun_marlboro

sadf

Change 26716 on 2002/05/07 by vromaker@vromaker_r400_sun_marlboro

asdf

Change 26713 on 2002/05/07 by vromaker@vromaker_r400_sun_marlboro

asdf

Change 26584 on 2002/05/06 by dougd@dougd_r400_sun_marlboro

added outputs o_vism_busy (to arbiter) and o_sp_vsr_read to shader pipe to control reading VSR during GPR loading. Removed input i_gpr_phase_mux as it was unused.

Change 26219 on 2002/05/03 by dougd@dougd_r400_sun_marlboro

initial submit for sq/vism rtl

Change 26218 on 2002/05/03 by dougd@dougd_r400_sun_marlboro

initial submit for sq/is rtl

Change 26217 on 2002/05/03 by dougd@dougd_r400_sun_marlboro

initial submit for sq/cfc rtl

Change 26216 on 2002/05/03 by dougd@dougd_r400_sun_marlboro

initial submit for sq/texconst rtl

Change 26214 on 2002/05/03 by dougd@dougd_r400_sun_marlboro

initial submit for sq/aluconst rtl

Change 26208 on 2002/05/03 by dougd@dougd_r400_sun_marlboro

intitial submit.

Change 25779 on 2002/05/01 by vromaker@vromaker_r400_sun_marlboro

latest updates

Change 25625 on 2002/04/30 by vromaker@vromaker_r400_sun_marlboro

updates

Change 25183 on 2002/04/26 by vromaker@vromaker_r400_sun_marlboro

file updates

Change 24711 on 2002/04/24 by vromaker@vromaker_r400_sun_marlboro

ping-pong buffer (ctl and storage, width parameterized)

Change 24469 on 2002/04/23 by vromaker@vromaker_r400_sun_marlboro

mux to select gfx register data based on state (context)

Change 24081 on 2002/04/19 by vromaker@vromaker_r400_sun_marlboro

initial versions

Change 23514 on 2002/04/16 by vromaker@vromaker_r400_sun_marlboro

updating with latest versions

Change 21716 on 2002/04/03 by vromaker@vromaker_r400_sun_marlboro

update

Change 21714 on 2002/04/03 by vromaker@vromaker_r400_sun_marlboro

update

Change 21642 on 2002/04/03 by vromaker@vromaker_r400_sun_marlboro

SP_TP_const to 48 bits

Change 21626 on 2002/04/03 by vromaker@vromaker_r400_sun_marlboro

latest fixes

Change 21468 on 2002/04/02 by vromaker@vromaker_r400_sun_marlboro

more fixes

Change 21425 on 2002/04/02 by vromaker@vromaker_r400_sun_marlboro

latest fixes

Change 21081 on 2002/03/29 by vromaker@vromaker_r400_sun_marlboro

Change 21079 on 2002/03/29 by vromaker@vromaker_r400_sun_marlboro

initial version

Change 21075 on 2002/03/29 by vromaker@vromaker_r400_sun_marlboro

initial version

Change 21074 on 2002/03/29 by vromaker@vromaker_r400_sun_marlboro

update

Change 21073 on 2002/03/29 by vromaker@vromaker_r400_sun_marlboro

initial version

Change 20660 on 2002/03/27 by vromaker@vromaker_r400_sun_marlboro

module name updated to sq

Change 20657 on 2002/03/27 by vromaker@vromaker_r400_sun_marlboro

position_space -> pos_avail, buffer_space -> buf_avail

Change 20655 on 2002/03/27 by vromaker@vromaker_r400_sun_marlboro

added SQ_TP_type, SQ_TP_send, un_TP_SQ_type, TP_SQ_rdy

Change 20654 on 2002/03/27 by vromaker@vromaker_r400_sun_marlboro

un_SQ_TP_pmask -> un_SQ_TP_pix_mask (for n = 0..3)

Change 20652 on 2002/03/27 by vromaker@vromaker_r400_sun_marlboro

latest version - renamed from sequencer_top.v

Change 19789 on 2002/03/20 by vromaker@vromaker_r400_sun_marlboro

re-added SQ_SP_ijline, fixed SP_TP instr and const widths

Change 19752 on 2002/03/20 by vromaker@vromaker_r400_sun_marlboro

put SQ_SP_stall back in

Change 19726 on 2002/03/20 by vromaker@vromaker_r400_sun_marlboro

updates

Change 19653 on 2002/03/19 by vromaker@vromaker_r400_sun_marlboro

updated sq top

Change 18260 on 2002/03/08 by vromaker@vromaker_r400_sun_marlboro

initial version

Change 18257 on 2002/03/08 by vromaker@vromaker_r400_sun_marlboro

initial version

Change 18256 on 2002/03/08 by vromaker@vromaker_r400_sun_marlboro

initial version

Change 11107 on 2001/12/03 by pmitchel@pmitchel_r400_win_marlboro

mv block dirs to gfx

Change 216876 on 2005/04/08 by vromaker@MA_VIC_P4

updated overview, removed some out-of-data info

Change 216874 on 2005/04/08 by vromaker@MA_VIC_P4

update

Change 191268 on 2004/10/12 by rramsey@rramsey_xenos_win_orl

Add a page for newCurCnt table

Change 188248 on 2004/09/17 by lseiler@lseiler_win_l_r400

Fixed a minor bug in the stencil function table

Change 149989 on 2004/02/19 by lseiler@lseiler_r400_win_marlboro1

Fixed bug in Zplane figure

Change 138566 on 2003/12/19 by fliljero@fl_frank

Added 3 new packets for improved type-0 packet processing:
Incremental_Update_State/Const/Instr

Change 137750 on 2003/12/16 by fliljero@fl_knarf

Added optimized Event_Write* packets & new opcodes

Change 137101 on 2003/12/12 by fliljero@fl_frank

Added Wait_Reg_Eq & Wait_Reg_Gte PM4 packet descriptions

Change 137025 on 2003/12/11 by fliljero@fl_knarf

updated documentation on error checking and removed reference to type-1 packet.

Change 136800 on 2003/12/10 by fliljero@fl_knarf

Updated description for MEM_WRITE_CNTR to include how to change the core clock interval from 1 <--> 16.

Change 136780 on 2003/12/10 by fliljero@fl_knarf

Updated Me_Init packet for Header Dumps & Error checking...added note about recompiling microcode to enable these debug only features.

Change 136762 on 2003/12/10 by fliljero@fl_knarf

Updates related to CP_MEQ

Change 136302 on 2003/12/08 by fliljero@fl_frank

Updates to MEQ related registers & busy signals

Change 135746 on 2003/12/05 by fliljero@fl_knarf

Updated CP Interrupt packet for performance

Change 134564 on 2003/12/01 by fliljero@fl_knarf

Max Buffer Size in Indirect Buffer Packets is [19:0]...Spec had [22:0]

Change 133990 on 2003/11/25 by jhoule@jhoule_doc_lt

v1.80 - Indicated that NO_ZERO srf mode is unsupported for Xenos (will currently only work in the VC path)

Change 133807 on 2003/11/25 by alleng@alleng_r400_win_marlboro_8200

Deleting old files...

Change 133806 on 2003/11/25 by alleng@alleng_r400_win_marlboro_8200

Deleted old files...

Change 133805 on 2003/11/25 by alleng@alleng_r400_win_marlboro_8200

Deleted old files...

Change 132833 on 2003/11/19 by fliljero@fl_knarf

changed R400 reference to Xenos

Change 131864 on 2003/11/13 by frising@frising_r400_win_marlboro

-For cube instruction SrcA swizzle is now .zzxy. Also tried to clarify the differences between what's shown in the numerics doc and what actually happens in the HW for cube instruction.

Change 130982 on 2003/11/10 by mpersaud@mpersaud_r400_win_tor

Submit delta doc for R400_R500 tvout changes

Change 130037 on 2003/11/04 by fliljero@fl_knarf

Added registers and PM4 packet changes related to the Software Managed Instruction Store...

Change 129511 on 2003/10/30 by tien@ma_spinach

Some additional info on perf and debug regs

Change 128832 on 2003/10/27 by tien@ma_spinach

Added info to perf and debug regs..

Change 127821 on 2003/10/22 by bbuchner@fl_bbuchner_r400_win

Updates

Change 127682 on 2003/10/21 by tien@ma_spinach

A little more one perf regs
New debug regs doc

Change 127599 on 2003/10/21 by tien@ma_spinach

Added some info the the perf counters for TP/TPC
This is tough :-) but kinda fun :-)

Change 127541 on 2003/10/21 by tien@ma_spinach

Filled in results for all cases
Added tri_juice cases
Added mip_frac = 0 case
Added z_frac = 0 case
Will define perf counters for TPC/TP here for the heck of it..

Change 126714 on 2003/10/15 by jayw@ma_jayw_lt

old update with John's change

Change 126588 on 2003/10/14 by tien@ma_spinach

Filled in numbers for a bunch of cases

Change 126058 on 2003/10/10 by frising@frising_r400_win_marlboro

-update scalar mova instructions to return MAX_S(SrcC)

Change 125972 on 2003/10/09 by frising@frising_r400_win_marlboro

-update vector mova instruction to be two operand with result to GPR being max of operands. Scalar mova instructions were updated to always return srcC.w.

Change 125952 on 2003/10/09 by beiwang@bei_pc

Added description and restrictions on RB->MH and MH->RB requests routed through MCCI.

Change 125904 on 2003/10/09 by jhoule@jhoule_doc_lt

v1.79

- Added stack map support
- Changed SIZE packing for 2D to allow for common decoding between stack maps and 2D maps
- Stated that SIZE values must contain w-1, h-1, and d-1
- Added "Stack" line to the maximum texture sizes

Change 125618 on 2003/10/08 by jiezhou@jiezhou_r400_win

small updating

Change 125614 on 2003/10/08 by jiezhou@jiezhou_r400_win

fix Hyperlink, add DTO description, Test counter description

Change 124923 on 2003/10/03 by jhoule@jhoule_doc_lt

v1.78

TFetchInstr:
- Removed unsupported opcodes for the sake of clarity

TFetchConst:
- Moved DIM field to last DWORD (kept the old one temporarily)
- Added ANISO_BIAS field

Formats:
- Added FMT_DXT3A_AS_1_1_1_1

Deprecated the ARBITRARY_FILTER fields from TFetch instr+const.

Change 124599 on 2003/10/02 by fliljero@fl_knarf

no change

Change 124344 on 2003/10/01 by lseiler@lseiler_r400_win_marlboro

Changes to depth formats to make HW more efficient

Change 124325 on 2003/10/01 by fliljero@fl_knarf

added 2nd interrupt from MC to RBBM

Change 124280 on 2003/10/01 by fliljero@fl_knarf

added MC0|MC1_RBBM_int signals

Change 123990 on 2003/09/30 by fliljero@fl_knarf

added changes to set_state and load_constant_context

Change 123796 on 2003/09/29 by vbhatia@vbhatia_r400_win_marlboro

Slight update of fmt49, to reflect changes in tp_fmt_encode hardware

Change 123793 on 2003/09/29 by tien@ma_spinach

First check-in

Change 123764 on 2003/09/29 by vgoel@fl_vgoel2

closed bug 104

Change 123315 on 2003/09/25 by fliljero@fl_knarf

Updated Const_Prefetch packet to issue only once per LCC packet. When the LCC ordinals repeat, they also repeat in the Const_Prefetch packet. Formerly, there was a new Const_Prefetch packet for each repeat of the ordinals.

Change 123064 on 2003/09/24 by fliljero@fl_knarf

Updated Subblk_Prefetch packet to send the Header only once, followed optionally by each ordinal on a mismatch.

Change 123059 on 2003/09/24 by ashishs@fl_ashishs_r400_win

closing bug 40 and 116

Change 123057 on 2003/09/24 by tien@ma_spinach

Upadted for the week 9/24

Change 123011 on 2003/09/24 by csampayo@fl_csampayo_r400

Closed bug #s 121, 123

Change 122955 on 2003/09/24 by mkelly@fl_mkelly_r400_win_laptop

Closed bugs 90 and 91...

Change 122800 on 2003/09/23 by fliljero@fl_knarf

made drawing change to reflect changes to SRC0 & SRC1 removal of MICROM, MRL & MRM as possible sources. also removed the BOOLEANs as a possible source for SRC1.

Change 122794 on 2003/09/23 by jhoule@jhoule_doc_lt

Update for the new Ws which has 11b mantissa (12b total)

Change 122741 on 2003/09/23 by csampayo@fl_csampayo_r400

Closed bug# 117. Some housekeeping

Change 122572 on 2003/09/22 by efong@efong_r400_win_tor_doc

added in dglen

Change 121971 on 2003/09/18 by efong@efong_r400_win_tor_doc

New update to remove people who have left and new PEYs

Change 121907 on 2003/09/17 by alleng@alleng_r400_win_marlboro_8200

Minor fixes, rearranged, added SC efficiency, vector ratios, etc...

Change 121820 on 2003/09/17 by vliu@vliu_r400_cnnbdv3_win_cvd

Initial revision

Change 121788 on 2003/09/17 by tien@ma_spinach

Updates for the week

Change 121752 on 2003/09/17 by koyu@kyu

added SQ spreadsheet, added -optimize to pm4opt redundant LCC and SET_CONST pkts

Change 121616 on 2003/09/16 by alleng@alleng_r400_win_marlboro_8200

Added new perl script to go direct from phantom.csv to the file.xls file.

Currently need to take this file, phantom_template.xls (in pv), and the phantom.csv and test_sum.txt files created by running the test in one directory and execute this script (perl perf2xls.pl).

Cannot be run on linux and requires the OLE32 perl module installed...

Change 121318 on 2003/09/15 by ctaylor@fl_ctaylor_r400_win_marlboro

Removed as these were redundant drawings.

Change 121306 on 2003/09/15 by vliu@vliu_r400_cnnbdv3_win_cvd

Test tiling library

Change 120795 on 2003/09/11 by fliljero@fl_knarf

added zpass_done info to the event_write packet

Change 120701 on 2003/09/11 by lkang@lkang_r400_win_tor

deletion

Change 120508 on 2003/09/10 by tien@ma_spinach

More updates afetr email from JOcelyn

Change 120486 on 2003/09/10 by tien@ma_spinach

Updates for the week

Change 120303 on 2003/09/09 by fliljero@fl_knarf

added predicated bin test results (RT/nRT) to State Management register w/index=0xD

Change 120271 on 2003/09/09 by fliljero@fl_knarf

Update Event Write packet for new functionality for the zpass_done event ... clears the context valid flag, which in turn will cause the context to be rolled on the next state packet.

Change 120048 on 2003/09/08 by jayw@ma_jayw_lt

changed pmask order in cache for 4-sample
John found better arrangement. one read for pmask and stencil
no ram line overlap.

Change 119978 on 2003/09/08 by fghodrat@ma_fghodrat

moved to xenos tree

Change 119939 on 2003/09/08 by fliljero@fl_knarf

added 128-bit write enable to the MH field to the CP_DEBUG register.

Change 119760 on 2003/09/05 by alleng@alleng_r400_win_marlboro_8200

Added new tests to pv_results
Added VGT and PA rates to the phantom template

Change 119726 on 2003/09/05 by fliljero@fl_knarf

added predicate_disable bit to CP_DEBUG

Change 119667 on 2003/09/05 by fliljero@fl_knarf

removed DATA ordinal from the MEM_WRITE_CNTR packet description

Change 119663 on 2003/09/05 by fliljero@fl_knarf

added MEM_WRITE_CNTR opcode
moved SET_BIN_MASK/SELECT opcodes to unused locations

Change 119540 on 2003/09/04 by tien@ma_spinach

UPdated some missing fields

Change 119483 on 2003/09/04 by frising@frising_r400_win_marlboro

v.1.77
-Added new compressed texture formats: FMT_DXT3A, FMT_DXT5A and FMT_CTX1 along with associated documentation.
-all these formats support degamma
-DXN now also supports degamma
-removed some cruft :)
-closed open question on supporting color keying

Change 119460 on 2003/09/04 by mkelly@fl_mkelly_r400_win_laptop

Branching example slides.

Change 119373 on 2003/09/04 by fghodrat@ma_fghodrat

update todo list

Change 119321 on 2003/09/03 by tien@ma_spinach

MOre changes

Change 119315 on 2003/09/03 by tien@ma_spinach

Something wacky with the clientspec, need to check in to re-update, plus some more updates form mtg.

Change 119301 on 2003/09/03 by fliljero@fl_knarf

made updates to the event write packet and added new associated register: CP_ME_CF_EVENT_SRC

Change 119277 on 2003/09/03 by fghodrat@ma_fghodrat

Rename cg_pm_r500.doc To cg_pm_xenos.doc

Change 119259 on 2003/09/03 by tien@ma_spinach

Updated for this week

Change 119253 on 2003/09/03 by bbloemer@ma_bbloemer

Added new document.

Change 119223 on 2003/09/03 by fliljero@fl_knarf

added CP_PROG_COUNTER,
related update to CP_ME_CNTL,
related update to EVENT_WRITE packet, &
related new PM4 packet MEM_WRITE_CNTR

Change 119196 on 2003/09/03 by fghodrat@ma_fghodrat

cg and pm spec for xenos

Change 118796 on 2003/08/29 by keli@keli_r400_win_tor

updates

Change 118786 on 2003/08/29 by keli@keli_r400_win_tor

Toronto Virage Memories Generation

Change 118771 on 2003/08/29 by llefebvr@llefebvr_r400_montreal

Fixing number of bits in the auto-count.

Change 118731 on 2003/08/29 by keli@keli_r400_win_tor

Document for Code coverage, formal verification, leda and synthesis report and web page generation

Change 118709 on 2003/08/29 by fliljero@fl_fliljeros

added real-time versions of the predicate registers: BIN_MASK & BIN_SELECT

Change 118570 on 2003/08/28 by kryan@kryan_r400_win_marlboro_DOCS

- Clean up

- Update some references and outdated facts.

Change 118408 on 2003/08/27 by fliljero@fl_knarf

updated/added coherency registers and interface
updated/added predicate registers and description

Change 118393 on 2003/08/27 by tien@ma_spinach

Updated the list
Merged c1 and non-c1 rtl tasks

Change 118362 on 2003/08/27 by fliljero@fl_knarf

added type-3 predicated packet related information

Change 117997 on 2003/08/25 by lkang@lkang_r400_win_tor

incremental update for physical partition

Change 117602 on 2003/08/21 by tien@ma_spinach

Updated

Change 117591 on 2003/08/21 by mkelly@fl_mkelly_r400_win_laptop

Slides for Perforce Branching presentation.

Change 117496 on 2003/08/21 by frising@frising_r400_win_marlboro

v.1.76
-changed polarity of INDEX_ROUND bit in vertex fetch instruction

Change 117394 on 2003/08/20 by tien@ma_spinach

Gradient task added

Change 117320 on 2003/08/20 by jyarasca@jyarasca_r400_win_cvd

Updated scheduling information on 247 Linux and 247 Chip Linux

Change 117312 on 2003/08/20 by tien@ma_spinach

...

Change 117236 on 2003/08/20 by tien@ma_spinach

Added some more tasks to list

Change 117002 on 2003/08/18 by tien@ma_spinach

Added to Perforce so I can edit it on multiple machines :-)

Change 116974 on 2003/08/18 by tien@ma_spinach

Fixed the encoding. 16_EXPAND are going to need their own :-)

Change 116968 on 2003/08/18 by tien@ma_spinach

Describes how DATA_FORMAT is encoded to reduce logic after walker.

Change 116959 on 2003/08/18 by tien@ma_spinach

Filled in more stuff for test list
Added to-do list

Change 116958 on 2003/08/18 by ctaylor@fl_ctaylor_r400_win_marlboro

Added SC block diagrams from Mike Mantor

Change 116957 on 2003/08/18 by jayw@MA_JAYW

updated pmask and stencil

Change 116952 on 2003/08/18 by beiwang@bei_pc

Added reminder for tPDEX test during Dynamic CKE test

Change 116866 on 2003/08/18 by jayw@ma_jayw_lt

no change

Change 116789 on 2003/08/15 by jayw@MA_JAYW

1 and 4 sample cache line arrangement updated

Change 116785 on 2003/08/15 by tmartin@tmartin_r400_win

added r400vc_fetch_mode_01 and r400vc_array_size_01

Change 116764 on 2003/08/15 by tien@ma_spinach

Added info

Change 116749 on 2003/08/15 by tien@ma_spinach

Added some info.

Change 116748 on 2003/08/15 by tien@ma_spinach

Adding...

Change 116699 on 2003/08/15 by jayw@ma_jayw_lt

visio bugs present

Change 116634 on 2003/08/14 by ashishs@fl_ashishs_r400_win

updated the tracker with 5 more ALU instruction tests. Also updated the total count on ALU instructions thereby increasing the project overall % complete

Change 116622 on 2003/08/14 by mkelly@fl_mkelly_r400_win_laptop

Update comment in _11
Copy _11 to _12 and use 144 vertices per packet
Update test_list and tracker accordingly.

Change 116617 on 2003/08/14 by csampayo@fl_csampayo_r400

Renamend sheet1, updated schedule

Change 116407 on 2003/08/13 by jasif@jasif_r400_win_tor

Made some additions. Will add some more tomorrow.

Change 116385 on 2003/08/13 by ygiang@ygiang_r400_win_marlboro_p4

updated:pv results

Change 116378 on 2003/08/13 by ygiang@ygiang_r400_win_marlboro_p4

updated: performance excel sheets

Change 116369 on 2003/08/13 by frising@frising_r400_win_marlboro

no changes, just a test.

Change 116347 on 2003/08/13 by jimmylau@jimmylau_r400_win_tor

Add a section on clock muxing conditions to the R500 BIF implementation specs.
Add a table of R500 pin and ROM straps.

Change 116338 on 2003/08/13 by jcox@FL_JCOX3

Make ready to post test plan status on web

Change 116194 on 2003/08/12 by tmartin@tmartin_r400_win

added r400vc_endian_swap_01 and r400vc_endian_swap_02

Change 115764 on 2003/08/11 by ashishs@fl_ashishs_r400_win

updated

Change 115683 on 2003/08/08 by koyu@kyuCA

added cycles/inst for vertex shader and pixel shader

Change 115607 on 2003/08/08 by mkelly@fl_mkelly_r400_win_laptop

Negative ALU VS constant clamping, negative index clamping with negative stepping

Change 115561 on 2003/08/08 by fliljero@fl_knarf

renamed references from R400 to Crayola

Change 115547 on 2003/08/08 by fliljero@fl_knarf

Removed all references to PIO/Push mode and its associated registers:
CP_CSQ_CNTL
CP_ 'RING | INDIRECT1 | INDIRECT2 | REAL_TIME | IB_ST | RT_ST'_PUSH

Change 115546 on 2003/08/08 by fliljero@fl_knarf

renamed to use Crayola rather than R400

Change 115480 on 2003/08/07 by mkelly@fl_mkelly_r400_win_laptop

First test of series which checks positive alu constant index clamping.

Change 115463 on 2003/08/07 by fliljero@fl_knarf

Baseline for the PM4 Spec (after the start of Xenos)

Change 115462 on 2003/08/07 by fliljero@fl_knarf

added note to cover to see PM4 Spec Crayola for the latest PM4 data

Change 115461 on 2003/08/07 by fliljero@fl_knarf

Baseline for Crayola CP Spec (after the start of Xenos)

Change 115460 on 2003/08/07 by fliljero@fl_knarf

added note on cover to see CP Spec Crayola for the latest CP data.

Change 115388 on 2003/08/07 by ashishs@fl_ashishs_r400_win2

updating the tracker for all the tests added in last week and current week

Change 115373 on 2003/08/07 by csampayo@fl_csampayo_r400

Updated status for the test r400sx_vtx_export_full_sequential_01

Change 115276 on 2003/08/06 by jhoule@jhoule_doc_lt

Changed the weights to give less pointy tents.
This forces a mutliplier instead of a shifter, but quality is deemed important enough to warrant those.

Change 115225 on 2003/08/06 by jhoule@jhoule_doc_lt

Document describing the new HiColor accumulation scheme

Change 115185 on 2003/08/06 by kevino@kevino_r400_win_marlboro

Updated document to reflect what is in RTL code for tca regs

Change 115176 on 2003/08/06 by mzhu@mzhu_crayola_win_tor

Add 3.4.9.21 for the cases right edge of icon/cursor is aligned with right edge of graphics window.

Change 115166 on 2003/08/06 by koyu@kyuCA

added new fields for SQ

Change 115088 on 2003/08/05 by jimmylau@jimmylau_r400_win_tor

rename scan ports from *BIF_* to *BIF_TOP_*

Change 114954 on 2003/08/05 by mkelly@fl_mkelly_r400_win_laptop

Add 3 simple tests

Change 114824 on 2003/08/04 by alleng@alleng_r400_win_marlboro_8200

Fixed hyperlinks

Change 114814 on 2003/08/04 by alleng@alleng_r400_win_marlboro_8200

Updates...

Change 114724 on 2003/08/04 by llefebvr@llefebvr_r400_montreal

Corrected the max number for mem exports to be 5 instead of 9.

Change 114564 on 2003/08/01 by aashkar@aashkar2_crayola_win

Updated Spec with the addition of bit 19 in the interrupt registers for the software interrupt (SW_INT). This interrupt is moving to the CP from the MH.

Change 114555 on 2003/08/01 by tmartin@tmartin_r400_win

fixed the total test count because some tests were left out

Change 114550 on 2003/08/01 by tmartin@tmartin_r400_win

added r400vc_fetch_mode_01 and r400vc_fetch_mode_02

Change 114343 on 2003/07/31 by csampayo@fl_csampayo_r400

Add memory export test, update test list and tracker

Change 114314 on 2003/07/31 by kryan@kryan_r400_win_marlboro_DOCS

Update with latest changes to Shader Assembler

- Update CUBE Vector ALU operation opcode syntax to take two source

  operands.

- VFETCH instruction modifications

  . Update offset field in VFETCH instruction to be 23 bit signed value

    instead of unsigned 8 bits from previous definition.

  . Modified syntax to add FETCH_TYPE (MEGA/MINI) and COUNT

    optional fields from Vfetch instruction.

Change 114266 on 2003/07/31 by tmartin@tmartin_r400_win

added r400vc_addr_spanning_01

Change 114220 on 2003/07/31 by jiezhou@jiezhou_r400_win

add description of fcp clock

Change 114162 on 2003/07/31 by alleng@alleng_r400_win_marlboro_8200

Run with updated hw

Change 114063 on 2003/07/30 by tmartin@tmartin_r400_win

moved the section of the clamping test

Change 114059 on 2003/07/30 by jhoule@jhoule_doc_lt

Hardcoded weights for anisotropy fix (not yet official).

Change 114044 on 2003/07/30 by csampayo@fl_csampayo_r400

Updated description of section 1.2.7

Change 114018 on 2003/07/30 by tmartin@tmartin_r400_win

added strides/offsets tests

Change 113992 on 2003/07/30 by csampayo@fl_csampayo_r400

Update individual requirements based on combined interaction

Change 113980 on 2003/07/30 by jimmylau@jimmylau_r400_win_tor

Remove ROM_strap_vcoref & ROM_strap_calref from the interface with strap block because they are shared with ROM_strap_pad_rx_manual_impedance & ROM_strap_pad_tx_manual_impedance.

Add ROM strap B_PRX_LBACK_EN, which shares with bit 0 of ROM strap PAD_CURRENT

Change 113956 on 2003/07/30 by jiezhou@jiezhou_r400_win

Initial release

Change 113883 on 2003/07/29 by alleng@alleng_r400_win_marlboro_8200

Included initial idle and busy counts.
Rearranged a bit...

Change 113792 on 2003/07/29 by csampayo@fl_csampayo_r400

Adjusted block schedules as per latest plan

Change 113761 on 2003/07/29 by mzhu@mzhu_crayola_win_tor

Test data clamping in test case 3 for fix point alpha format in 3.4.9.18 64bpp graphics with graphics and overlay alpha blending mode 1

Change 113507 on 2003/07/28 by csampayo@fl_csampayo_r400

Some housekeeping

Change 113483 on 2003/07/28 by ashishs@fl_ashishs_r400_win

updated

Change 113282 on 2003/07/25 by csampayo@fl_csampayo_r400

Updated status for tests r400sx_vtx_point_size_export_01-04 and added them to test_list

Change 113280 on 2003/07/25 by jayw@ma_jayw_lt

Working document for register read allocation across RB and DBs.

Change 113262 on 2003/07/25 by tmartin@tmartin_r400_win

no new tests just some updates

Change 113136 on 2003/07/25 by bbloemer@ma_bbloemer

Updated test descriptions.

Change 113130 on 2003/07/25 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 113128 on 2003/07/25 by kevino@kevino_r400_win_marlboro

updated tables with reg addresses

Change 112980 on 2003/07/24 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 112914 on 2003/07/24 by alleng@alleng_r400_win_marlboro_8200

Bringing the R400 docs up to date and checking in pm4play.bat

Change 112872 on 2003/07/24 by tmartin@tmartin_r400_win

added r400vc_base_addr_range_pci_01

Change 112683 on 2003/07/23 by chwang@chwang_r400_doc_win

Update.

Change 112655 on 2003/07/23 by kevino@kevino_r400_win_marlboro

Made table titles captions and updated table of tables

Change 112640 on 2003/07/23 by kevino@kevino_r400_win_marlboro

Added fetch gen and TCD debug tables

Change 112628 on 2003/07/23 by efong@efong_r400_win_tor_doc

updated visio diagrams

Change 112627 on 2003/07/23 by efong@efong_r400_win_tor_doc

Updated the test_control section of the document

Change 112623 on 2003/07/23 by jowang@jowang_R400_win

submit for kaleidoscope snapshot

Change 112603 on 2003/07/23 by jimmylau@jimmylau_r400_win_tor

Elaborate when the strap valid signals should be asserted in the ROM straps section.

Change 112518 on 2003/07/22 by tmartin@tmartin_r400_win

added r400vc_base_addr_range_agp_01

Change 112513 on 2003/07/22 by cbrennan@cbrennan_r400_win_marlboro

Allocated most of Ray's ports.
Added headings for more stuff to come.

Change 112496 on 2003/07/22 by brianf@ma_bfavela

Updated performance numbers with "better" architecture

Change 112470 on 2003/07/22 by paulv@MA_PVELLA

Fixed Table 22 to include the MH_TC_mcNsource bit.

Change 112465 on 2003/07/22 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 112329 on 2003/07/22 by jhoule@jhoule_doc_lt

Updated notes below in order to explain the 16.2 and 32.2 precision decision.

Change 112321 on 2003/07/22 by jhoule@jhoule_doc_lt

Updated with full channel separation, meaning that the 8b can now be OR'd together to create Mid and HiColor.

Change 112257 on 2003/07/21 by frising@frising_r400_win_marlboro

0.99n
-fixed small typo in CUBE instruction comments

Change 112232 on 2003/07/21 by tmartin@tmartin_r400_win

Added 5 tests. All test the dynamic addressing range of the VC when vertex buffers are stored in the frame buffer.

Change 112229 on 2003/07/21 by cbrennan@cbrennan_r400_win_marlboro

Add the beginnings of a TC debug registers document for review.

Change 112227 on 2003/07/21 by enewman@enewman_r400_linux_marlboro

fixed port_matcher command line switches and p4 label command line switches

Change 112113 on 2003/07/21 by rthambim@rthambim_r400_win_tor

Modified top level diagram and added comments.

Change 112100 on 2003/07/21 by frising@frising_r400_win_marlboro

v.1.75
-remove per-quad value for USE_REG_LOD since we can do it per-pixel full speed. Note that value 1 is now the only 'Yes'.

Change 112085 on 2003/07/21 by jyarasca@jyarasca_r400_win_cvd

Updated times

Change 112078 on 2003/07/21 by jimmylau@jimmylau_r400_win_tor

Updates on sections about slave interface changes and ROM strap location table, after specs review.

Change 112077 on 2003/07/21 by jasif@jasif_r400_win_tor

Updated schedules for simulation regressions.

Change 111975 on 2003/07/18 by csampayo@fl_csampayo_r400

Adding point size export mode test. Updated test_list and test tracker accordingly.

Change 111957 on 2003/07/18 by frising@frising_r400_win_marlboro

v.1.74
-Give a real explanation of how FMT_1_REVERSE differs from FMT_1

Change 111949 on 2003/07/18 by frising@frising_r400_win_marlboro

v.0.99m
-fix MAX4 instruction. Had comparision order backwards.

Change 111666 on 2003/07/17 by jimmylau@jimmylau_r400_win_tor

Minor changes to fix typos and to reword some paragraphs slightly.

Change 111558 on 2003/07/16 by jimmylau@jimmylau_r400_win_tor

Initial Revision

Change 111554 on 2003/07/16 by csampayo@fl_csampayo_r400

Some housekeeping

Change 111517 on 2003/07/16 by lseiler@lseiler_r400_win_marlboro

Minor fixes, additional test routines

Change 111482 on 2003/07/16 by tmartin@tmartin_r400_win

added r400vc_addr_alignment_01

Change 111413 on 2003/07/16 by smburu@smburu_r400_win_marlboro

tp_ch_blend update.

Change 111386 on 2003/07/16 by frising@frising_r400_win_marlboro

v.0.99l
-add scalar sin and cos instructions

Change 111385 on 2003/07/16 by frising@frising_r400_win_marlboro

v.1.99
-add scalar sin and cos instructions

Change 111285 on 2003/07/15 by gregs@gregs_r400_win_marlboro

typo in a signal name - corrected

Change 111281 on 2003/07/15 by brianf@ma_bfavela

More changes

Change 111229 on 2003/07/15 by brianf@ma_bfavela

Updated summary to include MH.

Change 111206 on 2003/07/15 by alleng@alleng_r400_win_marlboro_8200

Added a couple more tidbits regarding capture

Change 111200 on 2003/07/15 by brianf@ma_bfavela

Fixed hyperlinks

Change 111187 on 2003/07/15 by ashishs@fl_ashishs_r400_win

updated test tracker

Change 111168 on 2003/07/15 by paulv@MA_PVELLA

Updates concerning the MHS.

Change 111147 on 2003/07/15 by ashishs@fl_ashishs_r400_win

updated tracker

Change 111109 on 2003/07/15 by brianf@ma_bfavela

More performance updates

Change 111030 on 2003/07/14 by alleng@alleng_r400_win_marlboro_8200

Added more data to the results tab...

Change 110917 on 2003/07/14 by mzhu@mzhu_crayola_win_tor

Add 3.4.9.20 Multiply overlay alpha with global alpha for per pixel overlay alpha blend mode

Change 110885 on 2003/07/14 by brianf@ma_bfavela

Fixed DC so it doesn't error

Change 110883 on 2003/07/14 by brianf@ma_bfavela

Updated performance

Change 110825 on 2003/07/14 by jacarey@fl_jcarey2

Fix Typo in RBBM Spec Diagram

Change 110516 on 2003/07/11 by jiezhou@jiezhou_r400_win

Update PLL dividers' values

Change 110504 on 2003/07/11 by jiezhou@jiezhou_r400_win

update pll divider's value

Change 110337 on 2003/07/10 by vbhatia@vbhatia_r400_win_marlboro

Updated tp and vc path formatter status

Change 110255 on 2003/07/10 by lseiler@lseiler_r400_win_marlboro2

Minor text edits, updated pdf version

Change 110230 on 2003/07/10 by mzhu@mzhu_crayola_win_tor

Add Multiplying overlay alpha with global alpha in 11.10 Overlay Keyer. It is used for overlay per-pixel alpha blending mode.

Change 110173 on 2003/07/10 by dglen@dglen_r400

Deleted file
Superceded by R500 Display Colour Spaces.xls

Change 110159 on 2003/07/10 by dglen@dglen_r400

Spreadsheet for matrix, gamma and color conversions in R500 display path (DCP and TV out)

Change 109959 on 2003/07/09 by jimmylau@jimmylau_r400_win_tor

Update to the BIF slave interface specs after the review meeting

Change 109954 on 2003/07/09 by llefebvr@llefebvr_r400_montreal

Fixing VC table.

Change 109817 on 2003/07/08 by rthambim@rthambim_r400_win_tor

Fixed naming convention.

Change 109812 on 2003/07/08 by jhoule@jhoule_doc_lt

Major change, with left-alignment instead or right shifts.

Change 109715 on 2003/07/08 by rthambim@rthambim_r400_win_tor

Updated the spec with review feedback.

Change 109709 on 2003/07/08 by rthambim@rthambim_r400_win_tor

updated the spec with review feedback - included ordering info to read req; modified timing diags; added comments to unused ports.

Change 109670 on 2003/07/08 by jimmylau@jimmylau_r400_win_tor

Fix typo in the MH-BIF signal in the master specs

Change 109493 on 2003/07/07 by csampayo@fl_csampayo_r400

Some housekeeping

Change 109402 on 2003/07/06 by gregs@laptop1

...

Change 109352 on 2003/07/04 by jimmylau@jimmylau_r400_win_tor

Update the section on strap interface after the review meeting.

Change 109187 on 2003/07/03 by jowang@jowang_R400_win

Includes 30bpp for twin single and dual-link TMDS

Change 109175 on 2003/07/03 by ashishs@fl_ashishs_r400_win

updated test_list and trackers

Change 109106 on 2003/07/03 by alleng@alleng_r400_win_marlboro_8200

Removed one of the RB tabs

Change 109104 on 2003/07/03 by alleng@alleng_r400_win_marlboro_8200

Added a few minor updates from Ko...

Change 109094 on 2003/07/03 by moev@moev

updates to the status of the tests.

Change 108965 on 2003/07/02 by alleng@alleng_r400_win_marlboro_8200

Updated with specific registry settings for capture

Change 108939 on 2003/07/02 by ashishs@fl_ashishs_r400_win

corrected small error with SU

Change 108933 on 2003/07/02 by ashishs@fl_ashishs_r400_win

updated the tracker to include author for CL/VTE tests so that its easy for debugging the number of tests in each block

Change 108927 on 2003/07/02 by ashishs@fl_ashishs_r400_win

updated tracker to include this weeks PA tests

Change 108905 on 2003/07/02 by ygiang@ygiang_r400_win_marlboro_p4

added: Test list for Perfsuite performace tests

Change 108866 on 2003/07/02 by ashishs@fl_ashishs_r400_win

updated for some bugs

Change 108861 on 2003/07/02 by rthambim@rthambim_r400_win_tor

Added source/frequency information for clock signals.

Change 108824 on 2003/07/02 by jacarey@fl_jcarey_desktop

Update Min / Max functions in emulator to match hardware.
Hardware produces a 32-bit signed extended result of 16-bit comparision value.

Change 108746 on 2003/07/01 by smburu@smburu_r400_win_marlboro

tp_hicolor update.

Change 108740 on 2003/07/01 by paulv@MA_PVELLA

Fixed a mistake about the size of the RB queue. It is 2, not 4.

Change 108736 on 2003/07/01 by smburu@smburu_r400_win_marlboro

tp_hicolor update.

Change 108702 on 2003/07/01 by alleng@alleng_r400_win_marlboro_8200

Added RB(C1) tab, added tests to vtx tab, et al

Change 108692 on 2003/07/01 by georgev@devel_georgevh2_r400_win_marlboro

Added bsub run options

Change 108680 on 2003/07/01 by jacarey@fl_jcarey_desktop

Add section documenting CP Idling before writing certain control registers.

Change 108579 on 2003/06/30 by jasif@jasif_r400_win_tor

Updated.

Change 108520 on 2003/06/30 by ashishs@fl_ashishs_r400_win

updated the tracker

Change 108516 on 2003/06/30 by frising@frising_r400_win_marlboro

v.0.99k
-remove references to R400_TP_NAN

Change 108514 on 2003/06/30 by gregs@gregs_r400_win_marlboro

new

Change 108418 on 2003/06/27 by jiezhou@jiezhou_r400_win

major updating for DC split

Change 108407 on 2003/06/27 by frising@frising_r400_win_marlboro

-fixed small typo, no version bump

Change 108406 on 2003/06/27 by frising@frising_r400_win_marlboro

v.1.73
-Add FMT_32_32_32_FLOAT vertex only format
-remove fast float stuff
-remove TP NAN support

Change 108359 on 2003/06/27 by gregs@gregs_r400_win_marlboro

added CG_VC_pm_enb register bit

Change 108350 on 2003/06/27 by sbagshaw@sbagshaw

minor updates to contents & table of contents

Change 108338 on 2003/06/27 by frising@frising_r400_win_marlboro

v.1.98
-show scalar instructions SUB_CONST_0 and SUB_CONST_1 as negated adds to be consistent with other subtraction instructions.

Change 108325 on 2003/06/27 by frising@frising_r400_win_marlboro

v.0.99j
-misc updates and clean-up

Change 108322 on 2003/06/27 by rthambim@rthambim_r400_win_tor

Initial revision for R500.

Change 108317 on 2003/06/27 by brianf@ma_bfavela

MCMH Performance Spreadsheet

Change 108311 on 2003/06/27 by rthambim@rthambim_r400_win_tor

Added pci-express changes. Initial revision for R500.

Change 108284 on 2003/06/27 by gregs@gregs_r400_win_marlboro

fixed bug in PMESTBCLY macro

Change 108282 on 2003/06/27 by jiezhou@jiezhou_r400_win

add pclk as a slave mode for one-shot debug control

Change 108273 on 2003/06/27 by jimmylau@jimmylau_r400_win_tor

Update BIF slave interface specs for R500.

Change 108167 on 2003/06/26 by vbhatia@vbhatia_r400_win_marlboro

Update status for addresser and formatter

Change 108103 on 2003/06/26 by sbagshaw@sbagshaw

Programming Guide updated for R500.
Includes information on the Scaler

Change 107862 on 2003/06/25 by smburu@smburu_r400_win_marlboro

tp_hicolor update.

Change 107724 on 2003/06/24 by csampayo@fl_csampayo_r400

Add footer (this time saved update)

Change 107722 on 2003/06/24 by csampayo@fl_csampayo_r400

Added footer

Change 107709 on 2003/06/24 by frising@frising_r400_win_marlboro

0.99i
-fix typos with set* scalar instructions
-document correct behavior with not equal predicate instructions and NaNs

Change 107687 on 2003/06/24 by frising@frising_r400_win_marlboro

0.99h
-add cube instruction updates (two operand, etc)

Change 107637 on 2003/06/24 by moev@moev

update to current state of testing

Change 107634 on 2003/06/24 by bbuchner@fl_bbuchner_r400_win

updated slides

Change 107630 on 2003/06/24 by frising@frising_r400_win_marlboro

v.1.97
-update cube instruction to take two operands. Output produced in a different order too.

Change 107622 on 2003/06/24 by smburu@smburu_r400_win_marlboro

hicolor status update.

Change 107617 on 2003/06/24 by georgev@devel_georgevh2_r400_win_marlboro

Added "nodump" message to instructions.

Change 107537 on 2003/06/23 by georgev@devel_georgevh2_r400_win_marlboro

Added new way of doing things.

Change 107536 on 2003/06/23 by bbuchner@fl_bbuchner_r400_win

added review
updated block diagram of cache

Change 107479 on 2003/06/23 by bbuchner@fl_bbuchner_r400_win

added RP block diagrams for L1 and L2 request processing

Change 107419 on 2003/06/23 by ashishs@fl_ashishs_r400_win

updating trackers and test_list

Change 107282 on 2003/06/20 by moev@moev

Status as of 6/20/03

Change 107253 on 2003/06/20 by llefebvr@llefebvr_r400_montreal

Backup, no major changes.

Change 107197 on 2003/06/20 by vbhatia@vbhatia_r400_win_marlboro

Status update for formatter

Change 107080 on 2003/06/19 by bbuchner@fl_bbuchner_r400_win

ADDed MI block diagram and description. Minor fixes.

Change 106800 on 2003/06/18 by mkelly@fl_mkelly_r400_win_laptop

First VC test documented...

Change 106777 on 2003/06/18 by smburu@smburu_r400_win_marlboro

tp_hicolor status update.

Change 106696 on 2003/06/18 by smburu@smburu_r400_win_marlboro

Update of tp_hicolor tests.

Change 106645 on 2003/06/17 by csampayo@fl_csampayo_r400

Updated FC column to contain selected test counter

Change 106643 on 2003/06/17 by csampayo@fl_csampayo_r400

Added full chip column

Change 106642 on 2003/06/17 by csampayo@fl_csampayo_r400

Updated schedule, added full chip column

Change 106640 on 2003/06/17 by csampayo@fl_csampayo_r400

Added full chip column

Change 106634 on 2003/06/17 by csampayo@fl_csampayo_r400

Updated schedule, added full chip column

Change 106631 on 2003/06/17 by csampayo@fl_csampayo_r400

Updated schedule, added full chip column

Change 106623 on 2003/06/17 by georgev@devel_georgevh2_r400_win_marlboro

Added changes from TP meeting.

Change 106525 on 2003/06/17 by csampayo@fl_csampayo_r400

Update header

Change 106524 on 2003/06/17 by csampayo@fl_csampayo_r400

Update header

Change 106523 on 2003/06/17 by csampayo@fl_csampayo_r400

Update header take 2

Change 106521 on 2003/06/17 by csampayo@fl_csampayo_r400

Header update

Change 106519 on 2003/06/17 by csampayo@fl_csampayo_r400

Header update

Change 106516 on 2003/06/17 by csampayo@fl_csampayo_r400

Some housekeeping

Change 106509 on 2003/06/17 by csampayo@fl_csampayo_r400

Some housekeeping

Change 106493 on 2003/06/17 by csampayo@fl_csampayo_r400

Some housekeeping

Change 106436 on 2003/06/16 by ashishs@fl_ashishs_r400_win

minor updates

Change 106410 on 2003/06/16 by alleng@alleng_r400_win_marlboro_8200

Updates to standalone playback info

Change 106395 on 2003/06/16 by jasif@jasif_r400_win_tor

DAC Report.

Change 106329 on 2003/06/16 by alleng@alleng_r400_win_marlboro_8200

Initial submission

Change 106294 on 2003/06/16 by ashishs@fl_ashishs_r400_win

updated

Change 106291 on 2003/06/16 by ashishs@fl_ashishs_r400_win

updated

Change 105958 on 2003/06/12 by vbhatia@vbhatia_r400_win_marlboro

Updated weekly regression status for deriv and aniso

Change 105882 on 2003/06/12 by jbrady@jbrady_r400_win

Remove border color, add thread_type.

Change 105862 on 2003/06/12 by jbrady@jbrady_r400_win

Update to reflect current vcrg partitioning.
Added clamp module.

Change 105849 on 2003/06/12 by ashishs@fl_ashishs_r400_win

adding place holder for some of the tests that will be planned for later

Change 105838 on 2003/06/12 by ashishs@fl_ashishs_r400_win

added 2 bugs that were currently found in SQ waterfalling

Change 105834 on 2003/06/12 by alleng@alleng_r400_win_marlboro_8200

Added miscellaneous test, added results column for a couple of sheets

Change 105776 on 2003/06/12 by smburu@smburu_r400_win_marlboro

tp_hicolor status.

Change 105768 on 2003/06/12 by sbagshaw@sbagshaw

Presentation for R500 DCDO Debug bus included.
Changes started to R400 Debug Bus specification document to describe changes to debug required for DC/DO split.

Change 105676 on 2003/06/11 by csampayo@fl_csampayo_r400

Update tests reqs and to include more functionality

Change 105633 on 2003/06/11 by jowang@jowang_R400_win

programming guide: bypass mode only

Change 105615 on 2003/06/11 by beiwang@bei_pc

Added items left from line coverage.

Change 105476 on 2003/06/10 by csampayo@fl_csampayo_r400

Initial check in

Change 105436 on 2003/06/10 by bbuchner@fl_bbuchner_r400_win

removed null request from L2A FIFO when clamping is true

Change 105426 on 2003/06/10 by bbuchner@fl_bbuchner_r400_win

updated all external I/O
modified clamping

Change 105407 on 2003/06/10 by georgev@devel_georgevh2_r400_win_marlboro

Added things to check test descriptions.

Change 105349 on 2003/06/10 by ygiang@ygiang_r400_win_marlboro_p4

added: excel perf results template

Change 105347 on 2003/06/10 by mzhu@mzhu_crayola_win_tor

Add 3.4.9.19 data clamping in LUT PWL mode

Change 105335 on 2003/06/10 by ygiang@ygiang_r400_win_marlboro_p4

testing auto update for web

Change 105333 on 2003/06/10 by ygiang@ygiang_r400_win_marlboro_p4

removed auto update

Change 105310 on 2003/06/10 by moev@moev

Updated tested blocks

Change 105299 on 2003/06/10 by ygiang@ygiang_r400_win_marlboro_p4

updated:pv Xsheet

Change 105286 on 2003/06/10 by pmitchel@pmitchel_entire_depot_win

moving results to doc_lib/pv_results

Change 105058 on 2003/06/09 by csampayo@fl_csampayo_r400

Updated header. Some housekeeping

Change 105039 on 2003/06/09 by ashishs@fl_ashishs_r400_win

updated

Change 105017 on 2003/06/09 by mzhu@mzhu_crayola_win_tor

Update clamping and rounding for PWL LUT mode in 11.8

Change 104949 on 2003/06/09 by mzhu@mzhu_crayola_win_tor

Update floating point LUT fill pattern in 3.4.4.

Change 104946 on 2003/06/09 by bbloemer@ma_bbloemer

Updated test descriptions.

Change 104736 on 2003/06/06 by ashishs@fl_ashishs_r400_win

updated the tracker and test_list for the newly added 4 tests

Change 104633 on 2003/06/06 by jacarey@fl_jcarey_desktop

Document resetting of read registers to zero on reset.

Change 104629 on 2003/06/06 by paulv@MA_PVELLA

Updates to the MHS.

Change 104557 on 2003/06/06 by efong@efong_r400_win_tor_doc

Updated Linux machines assignments

Change 104473 on 2003/06/05 by jowang@jowang_R400_win

added/modified after test plan review

Change 104460 on 2003/06/05 by vbhatia@vbhatia_r400_win_marlboro

Update LodDeriv and LodAniso status of 06/05/03
Also added latest tcd status as mentioned by Kevin O.

Change 104360 on 2003/06/05 by ygiang@ygiang_r400_win_marlboro_p4

added: new pv sheet

Change 104355 on 2003/06/05 by ygiang@ygiang_r400_win_marlboro_p4

relocated: pef excel sheets

Change 104347 on 2003/06/05 by ygiang@ygiang_r400_win_marlboro_p4

relocating files

Change 104338 on 2003/06/05 by moev@moev

changed WSO connectivity to WSO_P

Change 104330 on 2003/06/05 by ygiang@ygiang_r400_win_marlboro_p4

fixed:link

Change 104325 on 2003/06/05 by ygiang@ygiang_r400_win_marlboro_p4

added: performance excel sheets to perforce

Change 104324 on 2003/06/05 by jacarey@fl_jcarey_desktop

Reset VS & PS De-alloc fifos when ME overwrites the *_Avail_Count counters

Change 104320 on 2003/06/05 by smburu@smburu_r400_win_marlboro

Updated status fot tp_hicolor.

Change 104298 on 2003/06/05 by kcorrell@kcorrell_r400_docs_marlboro_nb

documented change in DC tag definition - increased request sequence field

Change 104273 on 2003/06/05 by jmarsano@MA_JMARSANO

Added ATPG and DBIST sections

Change 104244 on 2003/06/05 by mzhu@mzhu_crayola_win_tor

Update alpha pattern in 3.4.9.18

Change 104156 on 2003/06/04 by ashishs@fl_ashishs_r400_win

updated the tracker

Change 104154 on 2003/06/04 by bbuchner@fl_bbuchner_r400_win

updated L1 and L2 request fifo contents

Change 104036 on 2003/06/04 by rfevreau@rfevreau_r400_win

Updates

Change 104014 on 2003/06/04 by jacarey@fl_jcarey_desktop

Documentation for new debug bit in the CP.

Change 103838 on 2003/06/03 by mzhu@mzhu_crayola_win_tor

Rename 2 LUT modes: 256-entry table mode and piece wise linear mode
Add clamping and rounding for PWL LUT mode

Change 103836 on 2003/06/03 by mzhu@mzhu_crayola_win_tor

Rename 2 LUT modes: 256-entry table mode and piece wise linear mode

Change 103818 on 2003/06/03 by jhoule@jhoule_doc_lt

Updated RF expand table.
Negative indices indicate added precision (8.8 -> 7..-8, 16.8 -> 15..-8, 32.8 -> 31..-8).

Change 103794 on 2003/06/03 by tien@ma_spinach

Added a litte bit more info

Change 103783 on 2003/06/03 by tien@ma_spinach

Initial Checkin

Change 103568 on 2003/06/02 by mzhu@mzhu_crayola_win_tor

Update 3.4.9.18 for 64bpp graphics with graphics and overlay alpha blending mode 1 (per pixel graphics alpha mode)

Change 103567 on 2003/06/02 by mzhu@mzhu_crayola_win_tor

Update for 64 bpp graphics bit depth and graphics/overlay blend using per-pixel alpha from graphics channel (chapter 11.9 Graphic Keyer).

Change 103433 on 2003/05/30 by ashishs@fl_ashishs_r400_win

updated

Change 103432 on 2003/05/30 by ashishs@fl_ashishs_r400_win

updated

Change 103380 on 2003/05/30 by omesh@ma_omesh

Mostly complete spreadsheet. What is missing is:
1) Tests yet to be written: HiC and RB Register Read tests.
2) The correct test writer information for some of the tests.
3) Filenames and testcases for some of Frank Hsien's tests.

However, I believe the counts are accurate of the existing tests.

Change 103349 on 2003/05/30 by khabbari@khabbari2_r400_win

test plan updated

Change 103343 on 2003/05/30 by gregs@gregs_r400_win_marlboro

added two signals for DC/DO debug bus connections

Change 103310 on 2003/05/30 by llefebvr@llefebvr_r400_montreal

Added comments about address register.

Change 103185 on 2003/05/29 by khabbari@khabbari2_r400_win

r500 test plan added

Change 103148 on 2003/05/29 by kmahler@kmahler_r400_doc_lib

Updates for Random Shader Generator

Change 103144 on 2003/05/29 by omesh@ma_omesh

Made some more updates to reflect current status. Still need to update Frank Hsien and Mark's tests into the spreadsheet, along with Hier Stencil and Hier Z tests. Other categories to add include the ZPASS counter tests, etc.

Change 103099 on 2003/05/29 by jacarey@fl_jcarey_desktop

Added oper=comp to document.

Change 103063 on 2003/05/29 by csampayo@fl_csampayo_r400

Updated status for r400sq_const_index_03.

Change 103060 on 2003/05/29 by csampayo@fl_csampayo_r400

Added max constant memory addrs reg indexing test. Updated test_list and test tracker accordingly.

Change 103041 on 2003/05/29 by ashishs@fl_ashishs_r400_win

updated some description as well as TBD's

Change 103035 on 2003/05/29 by ashishs@fl_ashishs_r400_win

added the r400sq_trunc_01 test bug

Change 103033 on 2003/05/29 by ashishs@fl_ashishs_r400_win

added r400sq_floor_01 test to the tracker

Change 103025 on 2003/05/29 by ashishs@fl_ashishs_r400_win

corrected the schedule. Had incorrectly put the new 7 tests under APril20 when they were supposed to be in May25

Change 103018 on 2003/05/29 by ashishs@fl_ashishs_r400_win

added 6 new tests to the tracker

Change 102980 on 2003/05/28 by jowang@jowang_R400_win

add dual-link TMDS tests (not complete)

Change 102957 on 2003/05/28 by gregs@gregs_r400_win_marlboro

new part

Change 102954 on 2003/05/28 by omesh@ma_omesh

Updated atleast some more of the RB Test plan spreadsheet. WIll try to finish by tommorow.

Change 102893 on 2003/05/28 by mdesai@MA_MDESAI

Added status for week ending 5/30

Change 102744 on 2003/05/27 by jiezhou@jiezhou_r400_win new fb divider slip test

Change 102708 on 2003/05/27 by smburu@smburu_r400_win_marlboro

First tp_ch_blend status report.

Change 102686 on 2003/05/27 by mpersaud@mpersaud_r400_win_tor

Rev 0.2 Mahendra Persaud
Date: May 27, 2003
Update test flow section plus other general clean up.

Change 102623 on 2003/05/26 by mpersaud@mpersaud_r400_win_tor

Added new tests for R500.

Change 102613 on 2003/05/26 by mpersaud@mpersaud_r400_win_tor

Initial Rev.

Change 102612 on 2003/05/26 by rfevreau@rfevreau_r400_win

Updated with due dates

Change 102610 on 2003/05/26 by jasif@jasif_r400_win_tor

Updated.

Change 102608 on 2003/05/26 by chwang@chwang_r400_doc_win

Latest update.

Change 102603 on 2003/05/26 by mpersaud@mpersaud_r400_win_tor

Initial Rev.

Change 102413 on 2003/05/23 by mzhu@mzhu_crayola_win_tor

Update 3.4.9.17 for 8K virtual desktop support

Change 102406 on 2003/05/23 by georgev@devel_georgevh2_r400_win_marlboro

Documented gnuzip feature for mcmh test bench.

Change 102376 on 2003/05/23 by enewman@enewman_r400_linux_marlboro

initial p4 submission

Change 102326 on 2003/05/23 by mdesai@MA_MDESAI

Added addresser section

Change 102323 on 2003/05/23 by imuskatb@imuskatb_r400_win_cnimuskatb

updated for R500

Change 102292 on 2003/05/23 by mzhu@mzhu_crayola_win_tor

Add GRPH/OVL_SURFACE_OFFSET programming in 3.4.9.17 for 8K virtual desktop support

Change 102205 on 2003/05/22 by mzhu@mzhu_crayola_win_tor

Add 3.4.9.17 for 8K virtual desktop support
Add 3.4.9.18 for 64bpp graphics with graphics and overlay alpha blending mode 1 (per pixel graphics alpha mode)

Change 102146 on 2003/05/22 by mkelly@fl_mkelly_r400_win_laptop

Final, 4 textures on an RT rectangle...

Change 102113 on 2003/05/22 by rfevreau@rfevreau_r400_win

Updates

Change 102018 on 2003/05/21 by csampayo@fl_csampayo_r400

Updated test description, test_list and test tracker

Change 102016 on 2003/05/21 by mzhu@mzhu_crayola_win_tor

Add support for 64 bpp graphics bit depth and graphics/overlay blend using per-pixel alpha from graphics channel (chapter 11.9 Graphic Keyer).
Increase GRPH/OVL_X/Y_END from 13 bits to 14 bits to support 8K virtual desktop

Change 102014 on 2003/05/21 by mzhu@mzhu_crayola_win_tor

Increase GRPH/OVL_X_END registers from 13 bits to 14 bits to support 8K virtual desktop
Add registers DxGRPH_16BIT_ALPHA_MODE and DxGRPH_16BIT_FIXED_ALPHA_RANGE for 64 bpp graphics bit depth and graphics/overlay blend using per-pixel alpha from graphics channel.

Change 102004 on 2003/05/21 by vbhatia@vbhatia_r400_win_marlboro

TP/TC Standalone testbenches weekly progress spreadsheet.

Change 101984 on 2003/05/21 by llefebvr@llefebvre_laptop_r400

more precisions on XYST generated register.

Change 101957 on 2003/05/21 by khabbari@khabbari2_r400_win

r500 changes

Change 101939 on 2003/05/21 by bbuchner@fl_bbuchner_r400_win

cleaned up interface names
added 8 Dword L1 changes

Change 101923 on 2003/05/21 by jacarey@fl_jcarey_desktop

Clarification that Isync flushing occurs only before the first "draw" packet after the transition.

Change 101917 on 2003/05/21 by mzhu@mzhu_crayola_win_tor

Add a new test (test 28) in chapter 3.4.8.4 for the case moving cursor hot spot from outside of cursor image size to inside and only capture the second frame.

Change 101892 on 2003/05/21 by imuskatb@imuskatb_r400_win_cnimuskatb

updated R500DVwork.xls

Change 101800 on 2003/05/20 by jacarey@fl_jcarey_desktop

Correction to Gradfill prim type for rectangles.

Change 101688 on 2003/05/19 by csampayo@fl_csampayo_r400

Added bug# 116

Change 101680 on 2003/05/19 by llefebvr@llefebvre_laptop_r400

added arbiter to TP/VC output control flow machines.

Change 101645 on 2003/05/19 by mmantor@FL_mmantorLT_r400_win

minor changes to the sq_vc interface drawing.

Change 101630 on 2003/05/19 by ashishs@fl_ashishs_r400_win

updated for the new perspective tests added...

Change 101625 on 2003/05/19 by jacarey@fl_jcarey_desktop

Miscellaneous Corrections to documents w.r.t. 2D Coherency Rectangle Updates.

Change 101602 on 2003/05/19 by bbuchner@fl_bbuchner_r400_win

added clamping logic

Change 101476 on 2003/05/17 by gregs@gregs_r400_win_cc

memory interface changes

Change 101380 on 2003/05/16 by jacarey@fl_jcarey_desktop

Clarifications to pre-write-timer and pre-write-limit usage.

Change 101350 on 2003/05/16 by bbuchner@fl_bbuchner_r400_win

update L2 drawing
update snoop signals

Change 101317 on 2003/05/16 by csampayo@fl_csampayo_r400

Update tests req counts

Change 101315 on 2003/05/16 by csampayo@fl_csampayo_r400

Update test req counts

Change 101307 on 2003/05/16 by csampayo@fl_csampayo_r400

Initial check-in

Change 101304 on 2003/05/16 by bbuchner@fl_bbuchner_r400_win

fixed L2 raddr and waddr signal names

Change 101278 on 2003/05/16 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 101259 on 2003/05/15 by ashishs@fl_ashishs_r400_win

updated for r400sq_trunc_01

Change 101256 on 2003/05/15 by csampayo@fl_csampayo_r400

Updated status for test r400sq_pressure_context_combo_01

Change 101247 on 2003/05/15 by gabarca@gabarca_crayola_win_cvd

Added clear to the new interface

Change 101236 on 2003/05/15 by jasif@jasif_r400_win_tor

Updated.

Change 101225 on 2003/05/15 by bbuchner@fl_bbuchner_r400_win

Added C1 changes, modifications to drawings.
Describe the RG index processor
add L1 drawing
add clamping interface

Change 101205 on 2003/05/15 by gregs@gregs_r400_win_marlboro

update

Change 101188 on 2003/05/15 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 101155 on 2003/05/15 by ashishs@fl_ashishs_r400_win

labeled myself in the owner column of some tests

Change 101137 on 2003/05/15 by csampayo@fl_csampayo_r400

Updated test status for tests:
r400sq_auto_wrapping_memories_01
r400sq_vs_memory_wrap_01
Sorted test_list

Change 101040 on 2003/05/14 by jasif@jasif_r400_win_tor

Added section on random delays

Change 101037 on 2003/05/14 by llefebvr@llefebvr_r400_montreal

Fixing the spec some more to match R500. Added some diagrams (SQ internals)

Change 101033 on 2003/05/14 by chwang@chwang_r400_doc_win

Formatting update.

Change 100995 on 2003/05/14 by jasif@jasif_r400_win_tor

Updated state diagrams and errors that the model will print.

Change 100993 on 2003/05/14 by chwang@chwang_r400_doc_win

TMDS update spec.

Change 100991 on 2003/05/14 by chwang@chwang_r400_doc_win

Updated.

Change 100975 on 2003/05/14 by ashishs@fl_ashishs_r400_win

updated the tracker and test_list

Change 100964 on 2003/05/14 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 100945 on 2003/05/14 by jacarey@fl_jcarey_desktop

Clarifications to Set_Constant and LCC packets w.r.t. write enables for each CONST_ID type.

Change 100888 on 2003/05/14 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 100832 on 2003/05/13 by jasif@jasif_r400_win_tor

Updated request state machine diagram.

Change 100802 on 2003/05/13 by jasif@jasif_r400_win_tor

Expanded forcible signals section.

Change 100704 on 2003/05/13 by frising@frising_r400_win_marlboro

v.1.72
-GRAD_EXP_ADJUST_H|V only exists in texture constant now.
-SetFIlter4Weights marked as not supported on r400
-Add 1024 bit option to REQUEST_SIZE for C1

Change 100627 on 2003/05/13 by kcorrell@kcorrell_r400_docs_marlboro_nb

Fixed info field width and vc field width in TC/MH interface tables

Change 100583 on 2003/05/12 by gabarca@gabarca_crayola_win_cvd

R500 interface changes

Change 100549 on 2003/05/12 by jacarey@fl_jcarey_desktop

Microcode Update for 2D surface coherency

Change 100274 on 2003/05/09 by jowang@jowang_R400_win

updated hot-plug detection logic to support dual-link

Change 100246 on 2003/05/09 by ashishs@fl_ashishs_r400_win

updated the excel tracker sheet and the test_list up-to-date

Change 100222 on 2003/05/09 by gregs@gregs_r400_win_marlboro

update

Change 100163 on 2003/05/09 by rramsey@RRAMSEY_P4_r400_win

drawing to describe instruction store address wrapping in the
control_flow_seq and target_instr_fetch blocks

Change 100159 on 2003/05/09 by jacarey@fl_jcarey_desktop

Document setting of bit 20 in 2D Booleans as Default_Sel

Change 100132 on 2003/05/09 by alleng@alleng_r400_win_marlboro_8200

Updated these two files with new information

Change 100004 on 2003/05/08 by llefebvr@llefebvr_r400_montreal

Some interface updates.

Change 99995 on 2003/05/08 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 99985 on 2003/05/08 by lseiler@lseiler_r400_win_marlboro2

New 2D tiling formats

Change 99957 on 2003/05/08 by lseiler@lseiler_r400_win_marlboro2

Update HierStencil description

Change 99927 on 2003/05/08 by moev@moev

Virage Star Memory System verification test plan (block level)

Change 99921 on 2003/05/08 by jmarsano@MA_JMARSANO

Added SAMPLE instruction to list

Change 99916 on 2003/05/08 by bbuchner@fl_bbuchner_r400_win

added L1/L2 request processing description
updated for 512 bit wide cache
updated external interface signals

Change 99909 on 2003/05/08 by mkelly@fl_mkelly_r400_win_laptop

Update with some more useful info on control flow looping...

Change 99782 on 2003/05/07 by mdoggett@mdoggett_r400_win_platypus

Format 22, 23 moved to 32BPP in format conversion table.
Interlace selection bit changed from Z0 to C0.
Format 40 moved with Format 11 and 12 in format conversion table.
SM4 removed.
Completed updates to R500 version of TC.
Many unlisted changes.

Change 99777 on 2003/05/07 by frising@frising_r400_win_marlboro

v.1.96
-update rules for masking with Color/Fog export.

Change 99536 on 2003/05/07 by csampayo@fl_csampayo_r400

Updated tests status

Change 99510 on 2003/05/07 by jacarey@fl_jcarey_desktop

Updates to document for usage of "flush done" flag in the microcode.

Change 99502 on 2003/05/07 by jmarsano@MA_JMARSANO

TST validation documents

Change 99471 on 2003/05/06 by csampayo@fl_csampayo_r400

Initial check-in

Change 99456 on 2003/05/06 by ashishs@fl_ashishs_r400_win

removed the modules from the excel sheet

Change 99452 on 2003/05/06 by frising@frising_r400_win_marlboro

v.1.95

-Update spec to show that masking of exports is allowed for all exports now (possible only exception fog - TBD).

Change 99445 on 2003/05/06 by ashishs@fl_ashishs_r400_win

removing the button from the sheet, since it wasn't enabled

Change 99433 on 2003/05/06 by csampayo@fl_csampayo_r400

Some housekeeping

Change 99386 on 2003/05/06 by jacarey@fl_jcarey_desktop

Update

Change 99385 on 2003/05/06 by mkelly@fl_mkelly_r400_win_laptop

Update comments, new tests checking RT Constant indexing...

Change 99380 on 2003/05/06 by jacarey@fl_jcarey_desktop

Fix for 2D Coherency (Flushing TC)

Change 99190 on 2003/05/05 by gregs@gregs_r400_win_marlboro

update

Change 99076 on 2003/05/05 by kcorrell@kcorrell_r400_docs_marlboro_nb

First update for R500. Includes changes to AIC and TC interface for R500 feature support.

Change 99068 on 2003/05/05 by mkelly@fl_mkelly_r400_win_laptop

SQ RT constants and flow control testing

Change 98890 on 2003/05/02 by gregs@gregs_r400_win_marlboro

names changes again ..

Change 98760 on 2003/05/02 by llefebvr@llefebvr_r400_montreal

forgot to remove 1 waterfall signal.

Change 98670 on 2003/05/01 by jbrady@jbrady_r400_win

Add signals to SQ interface for flow control, count_lo, fetch_type. Add L1_request signal first_instr_of_vv.

Change 98628 on 2003/05/01 by frising@frising_r400_win_marlboro

0.99g
-fixed typo in float16<->float32 conversion table

Change 98626 on 2003/05/01 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 98544 on 2003/05/01 by mkelly@fl_mkelly_r400_win_laptop

Added detail on loop

Change 98539 on 2003/05/01 by kmahler@kmahler_r400_doc_lib

Some minor updates.

Change 98518 on 2003/05/01 by enewman@enewman_r400_linux_marlboro

fixed typos, cleaned up some stuff

Change 98500 on 2003/05/01 by llefebvr@llefebvr_r400_montreal

Refreshing the interfaces per Andi's last mail.

Change 98409 on 2003/04/30 by gregs@gregs_r400_win_marlboro

update

Change 98401 on 2003/04/30 by llefebvr@llefebvr_r400_montreal

Updated the SQ->SP interfaces for the R500.

Change 98394 on 2003/04/30 by bbloemer@ma_bbloemer

Updated description of DRAM software command unit.

Change 98305 on 2003/04/30 by moev@moev

Updated Tile Enable control register to match verilog

Change 98296 on 2003/04/30 by jayw@ma_jayw_lt

minor updates

Change 98285 on 2003/04/30 by fliljero@fl_frank

latest updates

Change 98162 on 2003/04/29 by mkelly@fl_mkelly_r400_win_laptop

Maximum pixel shader nested, control flow subroutines in RTS, with
non-RTS in front and back containing simple pixel and vertex shaders.

Updated SQ doc with RTS tests needed.

Change 97906 on 2003/04/28 by ashishs@fl_ashishs_r400_win

added a bug for the 2 failing CL tests

Change 97768 on 2003/04/25 by gregs@laptop1

all names on DC interface changed.

Change 97727 on 2003/04/25 by mzhu@mzhu_crayola_win_tor

Update 11.2.1 DCP Window Controller for DCP_DMIF_SIZE

Change 97706 on 2003/04/25 by jacarey@fl_jcarey_desktop

Add RBBM_DB_soft_reset to RBBM

Change 97673 on 2003/04/25 by mpersaud@mpersaud_r400_win_tor

Rev 0.06 Mahendra Persaud
Date: April 25, 2003
Expanded CLIENT_DCCIF_wc?_reg_align64byte to 2 bits

Change 97672 on 2003/04/25 by mpersaud@mpersaud_r400_win_tor

Rev 0.02 Mahendra Persaud
Date: April 25, 2003
Updated CLIENT_DCCIF_reg_wc?_align64byte functionality.
Expanded on some block descriptions
Updated diagrams

Change 97636 on 2003/04/25 by bbuchner@fl_bbuchner_r400_win

added L2 cache drawing
fixed cache tag size (stored) to 22 bits
fixed memory request to include 26 bits of address and sec. mask

Change 97613 on 2003/04/25 by khabbari@khabbari2_r400_win

not_last_line_pair is added

Change 97525 on 2003/04/24 by lkang@lkang_r400_win_tor

dc split update

Change 97523 on 2003/04/24 by bbuchner@fl_bbuchner_r400_win

added top level cache controller

Change 97450 on 2003/04/24 by llefebvr@llefebvre_laptop_r400

Updated stall conditions.
Made swizzle changes.
Added more R500 specifics.

Change 97441 on 2003/04/24 by mpersaud@mpersaud_r400_win_tor

Rev 0.01 Mahendra Persaud
Date: April 24, 2003
Initial revision.

Change 97440 on 2003/04/24 by mpersaud@mpersaud_r400_win_tor

Rev 0.05 Mahendra Persaud
Date: April 24, 2003
Added CLIENT_DCCIF_wc?_reg_align64byte signal to interface.
Fixed some signal descriptions and updated some of the timing diagrams

Change 97403 on 2003/04/24 by jacarey@fl_jcarey_desktop

RBBM Document Updates for DB

Change 97280 on 2003/04/23 by jowang@jowang_R400_win

try to update diagram so that it looks nice after rotate.
didn't work.

Change 97278 on 2003/04/23 by jowang@jowang_R400_win

modified after design review 04/23/03

Change 97277 on 2003/04/23 by jowang@jowang_R400_win

updated after the design review 04/23/03

Change 97272 on 2003/04/23 by frising@frising_r400_win_marlboro

Spazzed out on that last check-in. Should have read that 1bpp textures are filterable. No version bump.

Change 97261 on 2003/04/23 by frising@frising_r400_win_marlboro

v.1.71
-show that 1D textures formats are filterable
-remove REQUEST_LATENCY fields
-misc clean-up

Change 97254 on 2003/04/23 by bbuchner@fl_bbuchner_r400_win

added Request Generator Drawing

Change 97181 on 2003/04/23 by bbuchner@fl_bbuchner_r400_win

0.1 of Vertex cache document

Change 97161 on 2003/04/23 by llefebvr@llefebvre_laptop_r400

interface name changes for the SQ->SP fetch swizzles.

Change 97140 on 2003/04/23 by csampayo@fl_csampayo_r400

Update using Laurent's inputs

Change 97092 on 2003/04/23 by llefebvr@llefebvre_laptop_r400

Added SP stall conditions to the SQ spec.

Change 97084 on 2003/04/23 by mpersaud@mpersaud_r400_win_tor

Rev 0.04 Mahendra Persaud
Date: April 23, 2003
Fixed some of the port names to make them more consistent.

Change 97077 on 2003/04/23 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 97063 on 2003/04/23 by kcorrell@kcorrell_r400_docs_marlboro_nb

update MH_TCn_rtr signal description

Change 97033 on 2003/04/22 by frising@frising_r400_win_marlboro

v.1.94
-update mova* instructions to return SrcA (like a mov) on the vector side and SrcC.W replicated on the scalar side.
-update pred* instructions to only use W channel of operands.
-update GPR write-back table to show that scalar component is used when both scalar and vector write masks are enabled.

Change 97032 on 2003/04/22 by frising@frising_r400_win_marlboro

v.0.99f
-misc updates/corrections/clean-up

Change 97009 on 2003/04/22 by lchen@lchen_crayola0

update

Change 96906 on 2003/04/22 by bbuchner@fl_bbuchner_r400_win

VC DOCS

Change 96888 on 2003/04/22 by jasif@jasif_r400_win_tor

Keep track of R500 DV work.

Change 96884 on 2003/04/22 by jasif@jasif_r400_win_tor

Keep track of outstanding DV work.

Change 96881 on 2003/04/22 by jiezhou@jiezhou_r400_win

dual link tmds clock

Change 96875 on 2003/04/22 by fliljero@fl_frank

changed data from one pass to the next to better insure proper validation

Change 96872 on 2003/04/22 by jiezhou@jiezhou_r400_win

delete old diagram

Change 96860 on 2003/04/22 by mpersaud@mpersaud_r400_win_tor

New delta doc.
Updated test list.

Change 96854 on 2003/04/22 by mpersaud@mpersaud_r400_win_tor

Rev 0.03 Mahendra Persaud
Date: April 21, 2003
Update after interface review.

- general description cleanup
- update interface names
- moved surface number to address bits [5:4] during surface register write
- added functionality to rcd signal(not wired to zero anymore)
- removed client id's from read return interface.
- update timing diagrams

Change 96851 on 2003/04/22 by jacarey@fl_jcarey_desktop

1. Add detection of Type-0/1 Packets in IBs if Enabled in ME_INIT Packet.
2. Unit-Level Test Added to verify.
3. Update to PM4 Spec to document addition.

Change 96715 on 2003/04/21 by jowang@jowang_R400_win

Rotate the diagram by 90 degrees for r500_tmds_dual_link.doc

Change 96713 on 2003/04/21 by jowang@jowang_R400_win

updated dataSynchronizer document for R500

Change 96699 on 2003/04/21 by mkelly@fl_mkelly_r400_win_laptop

Test all 32 RTS boolean bits in the pixel shader...

Change 96592 on 2003/04/20 by gregs@gregs_r400_win_cc

update

Change 96572 on 2003/04/19 by llefebvr@llefebvr_r400_montreal

Documentation changes for R500.

Change 96524 on 2003/04/18 by jhoule@jhoule_doc_lt

Changed GetCompTexLOD and SetTexLOD opcodes to work with a single LOD component.
Removed 1D restriction for FMT_1* formats.
Indicated that arbitrary filters are now unsupported.
Realigned every row with auto-fit to fix incomplete last lines.

Change 96500 on 2003/04/18 by fliljero@fl_frank

latest updates

Change 96403 on 2003/04/18 by jacarey@fl_jcarey_desktop

Type 0/1 Error Checking in IBs (ME_INIT, Interrupt Registers)
Un-Link Diagrams from PM4 Spec

Change 96286 on 2003/04/17 by jacarey@fl_jcarey_desktop

1. Add INVALID_TAG to bit 31 of CP_MIU_TAG_STAT2 register
2. Added enumeration for the perfomance counter selects in the CP and RBBM

Change 96284 on 2003/04/17 by georgev@devel_georgevh2_r400_win_marlboro

Added MCMH coverage.

Change 96130 on 2003/04/17 by mpersaud@mpersaud_r400_win_tor

Update port names to match R500 naming convention.

Change 96064 on 2003/04/16 by gregs@gregs_r400_win_marlboro

update

Change 95964 on 2003/04/16 by jacarey@fl_jcarey_desktop

Add new debug registers that record when read tags are outstanding to the CP.

Change 95946 on 2003/04/16 by jacarey@fl_jcarey2

Removed Snooping Connections to DMA Engine from Diagram

Change 95802 on 2003/04/16 by mpersaud@mpersaud_r400_win_tor

Add to source control

Change 95773 on 2003/04/15 by jowang@jowang_R400_win

First rev of dual link TMDS block diagram

Change 95760 on 2003/04/15 by sbagshaw@sbagshaw

added detail that power management for DC block must be enabled to use "one shot" clock or clock branch stopping feature of test debug circuitry

Change 95740 on 2003/04/15 by gregs@gregs_r400_win_marlboro

update

Change 95666 on 2003/04/15 by csampayo@fl_csampayo_r400

Some housekeeping

Change 95655 on 2003/04/15 by csampayo@fl_csampayo_r400

Add "Owner" column

Change 95625 on 2003/04/15 by jacarey@fl_jcarey2

Updates to Document

Change 95606 on 2003/04/15 by jacarey@fl_jcarey2

Baseline of Ideas for Pre-emptive Ring Hardware in CP

Change 95542 on 2003/04/14 by sbagshaw@sbagshaw

Changed references to PIXCLK to refer to pixel PLL source clock branch instead of a particular display controller's pixel clock branch.
Changed all references of DISP1_PCLK to PIX1CLK.
Changed all references of DISP2_PCLK to PIX2CLK.
All prior references to "Primary display controller pixel clock" changed to "Primary pixel PLL source clock". Similarly, all prior references to "Secondary display controller pixel clock" changed to "Secondary pixel PLL source clock".

Change 95512 on 2003/04/14 by jacarey@fl_jcarey2

Miscellaneous Updates

Change 95471 on 2003/04/14 by jacarey@fl_jcarey2

Added some packet restrictions for BitBlt and HostDataBlt:

For HostData_Blt: Never identify a brush even though the ROP code is set to 0xCC

For BitBlt:Never do a simple BitBlt with a mono opaque source, SRC_TYPE=0, or a mono transparent source, SRC_TYPE=1, and a ROP code set to source copy, 0xCC.

Change 95460 on 2003/04/14 by frising@frising_r400_win_marlboro

v.1.69
-Add fields to vertex fetch instruction to support mega/mini fetches.

Change 95443 on 2003/04/14 by jasif@jasif_r400_win_tor

Fixed section on integration.

Change 95223 on 2003/04/11 by gregs@gregs_r400_win_marlboro

update

Change 95214 on 2003/04/11 by jacarey@fl_jcarey2

Baseline CP Review Slides

Change 95139 on 2003/04/11 by jiezhou@jiezhou_r400_win

for sclk_r_vga_rst

Change 95109 on 2003/04/11 by mkelly@fl_mkelly_r400_win_laptop

Test smallest Z offset and scale to produce a discernable difference
in the Zbuffer (1 lsb).

Change 95068 on 2003/04/10 by gregs@laptop1

update

Change 95039 on 2003/04/10 by gabarca@gabarca_crayola_win_cvd

Fixed case c6 of crtc display parameters tests

Change 95022 on 2003/04/10 by khabbari@khabbari2_r400_win

r500 changes

Change 94731 on 2003/04/09 by rfevreau@rfevreau_r400_win

Took out dispout_gpios from block level regression

Change 94703 on 2003/04/09 by jasif@jasif_r400_win_tor

Updated.

Change 94683 on 2003/04/09 by jasif@jasif_r400_win_tor

Updated

Change 94680 on 2003/04/09 by jasif@jasif_r400_win_tor

Added section on managing devel and dcsplit branch.

Change 94556 on 2003/04/08 by gregs@gregs_r400_win_marlboro

update

Change 94544 on 2003/04/08 by rfevreau@rfevreau_r400_win

Updates to xls files and Makefile

Change 94458 on 2003/04/08 by jasif@jasif_r400_win_tor

Describes how to use perforce branching mechanism for dc split changes.

Change 94419 on 2003/04/08 by jacarey@fl_jcarey2

Proposal #2 for test interface

Change 94395 on 2003/04/08 by rramsey@RRAMSEY_P4_r400_win

update

Change 94342 on 2003/04/07 by sbagshaw@sbagshaw

Added new test, "DISPOUT_INTERFACES_powerstate", to verify the Display Output data output interfaces function properly in different power states.

Change 94201 on 2003/04/07 by jacarey@fl_jcarey2

Test Bus Proposal

Change 94195 on 2003/04/07 by ashishs@fl_ashishs_r400_win

closed a bug

Change 94183 on 2003/04/07 by vgoel@fl_vgoel2

updated bug status

Change 94179 on 2003/04/07 by ashishs@fl_ashishs_r400_win

updated

Change 94030 on 2003/04/04 by paulv@MA_PVELLA

Updates to the MHS section.

Change 93969 on 2003/04/04 by gregs@gregs_r400_win_marlboro

update

Change 93958 on 2003/04/04 by sbagshaw@sbagshaw

R400 DC Test Debug document put into proper documentation template. Section explaining how to specify clock domain for each bit of DC_TEST_DEBUG_DATA added.

Change 93888 on 2003/04/04 by jacarey@fl_jcarey_desktop

Updated Documentation for addition of Pre-Fetch Matching for Loop and Boolean Constants

Change 93733 on 2003/04/03 by vgoel@fl_vgoel2

closed a bug

Change 93664 on 2003/04/03 by fliljero@fl_frank

latest passing updates

Change 93520 on 2003/04/02 by jowang@jowang_R400_win

no dithering done in blank pixel for DVOA

Change 93501 on 2003/04/02 by ashishs@fl_ashishs_r400_win

updated

Change 93481 on 2003/04/02 by csampayo@fl_csampayo_r400

Initial check-in

Change 93465 on 2003/04/02 by jacarey@fl_jcarey_desktop

1. Move RB_CLRCMP_MSK_HI and RB_CLRCMP_DST_HI initialization for 2D to the 2D Indirect Buffer
2. Mask for RB_CLRCMP_MSK_LO is dependant on the pixel type
3. Updated PM4 Spec Accordingly
4. Updated CP Unit-Level Tests Accordingly

Change 93456 on 2003/04/02 by llefebvr@llefebvr_r400_montreal

update to the control flow instruction.
Adding timing diagram for the SQ->VC/TP transfers.

Change 93409 on 2003/04/02 by ygiang@ygiang_r400_win_marlboro_p4

updated sp test excel sheet

Change 93373 on 2003/04/02 by ashishs@fl_ashishs_r400_win

filed bug for SQ tests failing due to Laurent's change # 92966

Change 93269 on 2003/04/01 by gabarca@gabarca_crayola_win_cvd

fixed horz parameters cases c8, c9, still don't know why c7 does not emulate properly

Change 93211 on 2003/04/01 by paulv@MA_PVELLA

Initial version.

Change 93199 on 2003/04/01 by gregs@gregs_r400_win_marlboro

update

Change 93176 on 2003/04/01 by gregs@gregs_r400_win_marlboro

first power estimate

Change 93130 on 2003/04/01 by nbarbier@nbarbier_r400_win_tor

Updated Power Management Section.

Change 93121 on 2003/04/01 by vgoel@fl_vgoel2

updated to remove bug 788

Change 93110 on 2003/04/01 by jiezhou@jiezhou_r400_win

updating table

Change 93065 on 2003/04/01 by jiezhou@jiezhou_r400_win

updating in pclk src selection

Change 92948 on 2003/03/31 by jiezhou@jiezhou_r400_win

take debug_test out from test plan.

Change 92850 on 2003/03/31 by efong@efong_r400_win_tor_doc

Added in Verdi Training docs

Change 92846 on 2003/03/31 by ashishs@fl_ashishs_r400_win

updated

Change 92637 on 2003/03/28 by fliljero@fl_frank

added dummy writes to instruction memory so that it does not return unknowns when read

Change 92627 on 2003/03/28 by alleng@alleng_r400_win_marlboro_8200

Initial submit

Change 92587 on 2003/03/28 by llefebvr@llefebvre_laptop_r400

added swizzle codes to the spec.

Change 92433 on 2003/03/27 by mpersaud@mpersaud_r400_win_tor

Rev 0.98 Mahendra Persaud
Date: March 26, 2003
Update section 4.1 with new display parameters and formulas.

Change 92425 on 2003/03/27 by jiezhou@jiezhou_r400_win

initial release

Change 92381 on 2003/03/27 by dwong@cndwong2

Added CP_RBBM_dma_busy to Bit 2 of the RT discrete signals

Change 92378 on 2003/03/27 by mkelly@fl_mkelly_r400_win_laptop

Add test requirement...

Change 92370 on 2003/03/27 by jacarey@fl_jcarey_desktop

Fix typo in RB_BUFSZ equation in the CP_RB_CNTL register

Change 92347 on 2003/03/27 by jiezhou@jiezhou_r400_win

update

Change 92330 on 2003/03/27 by ashishs@fl_ashishs_r400_win

filed bug for the hang caused by Laurent's change #92184

Change 92325 on 2003/03/27 by kcorrell@kcorrell_r400_docs_marlboro_nb

updated address translation diagram for rom reads, removed summary of registers in favor of a reference to the block file

Change 92272 on 2003/03/26 by rfevreau@rfevreau_r400_win

Updated numbers

Change 92265 on 2003/03/26 by jimmylau@jimmylau_r400_win_tor

Initial revision of BIF test plan.

Change 92238 on 2003/03/26 by jiezhou@cn_jiezhou

updating

Change 92204 on 2003/03/26 by ashishs@fl_ashishs_r400_win

documented the SQ bug by Laurent

Change 92175 on 2003/03/26 by gregs@laptop1

reverted ROM_AP_SIZE definition

Change 92174 on 2003/03/26 by mkelly@fl_mkelly_r400_win_laptop

Update with SC test coverage...

Change 92165 on 2003/03/26 by jacarey@fl_jcarey_desktop

Visio Updates to Scratch Register Interrupt Function in the CP

Change 92090 on 2003/03/25 by rfevreau@rfevreau_r400_win

Submitted

Change 92030 on 2003/03/25 by abeaudin@abeaudin_r400_win_marlboro

added build sequence

Change 91961 on 2003/03/25 by bbloemer@ma_bbloemer

Updated test plan with more test description and effort estimate.

Change 91878 on 2003/03/24 by jacarey@fl_jcarey_desktop

Add Test_Select to R400 documentation for the scratch register compare interrupt

Change 91821 on 2003/03/24 by jacarey@fl_jcarey_desktop

Scratch Register Interrupt

Change 91812 on 2003/03/24 by mzhu@mzhu_crayola_win_tor

Double MCLK frequency to reduce MH latency to less than 40% of HTOTAL for MH_DC_LATENCY1 test
Set SCLK to 440MHz for all MH_DMIF_DCP tests
Force dmif model rts delay signals to 0 on block level simulation for all MH_DMIF_DCP tests

Change 91806 on 2003/03/24 by jacarey@fl_jcarey_desktop

Fix typo in section 7.2

Change 91790 on 2003/03/24 by ashishs@fl_ashishs_r400_win

updated

Change 91783 on 2003/03/24 by kcorrell@kcorrell_r400_docs_marlboro_nb

Update to reflect changes to address decision tree and that primary target is now pci-express

Change 91625 on 2003/03/21 by nbarbier@nbarbier_r400_win_tor

Made additional change to tmds hpd override test.

Change 91582 on 2003/03/21 by gregs@gregs_r400_win_marlboro

various minor changes

Change 91540 on 2003/03/21 by jacarey@fl_jcarey2

Scratch Compare Interrupt Diagram for R400

This is the same logic that is being added for R390 as requested
by Jeffrey Cheng. The difference from R390 is the location of the
interrupt control and status bits. This is indicated on the diagram.

Change 91514 on 2003/03/21 by abeaudin@abeaudin_r400_win_marlboro

added directory description

Change 91495 on 2003/03/21 by csampayo@fl_csampayo_r400

Updated status for the following tests:
r400sq_flow_control_02
r400sq_flow_control_03

Change 91489 on 2003/03/21 by csampayo@fl_csampayo_r400

Some housekeeping

Change 91474 on 2003/03/21 by jacarey@fl_jcarey_desktop

1. Removed PREFETCH_DISABLE_OVERRIDE from CP_DEBUG register in CP Spec.
2. Updated PFP pseudocode for Indirect_Buffer and Indirect_Buffer_PFD packets.

Change 91438 on 2003/03/20 by csampayo@fl_csampayo3

VGT output path stress tests

Change 91354 on 2003/03/20 by dwong@cndwong2

include descriptions on FDCT compression

Change 91347 on 2003/03/20 by mdoggett@mdoggett_r400_win_platypus

Major change to cache design. Partial spec update to new design.
L1 removed, L2 replaced with 4 read port 128 word memories.
Added new Set, Halfline, Slice creation in Cacheline Formats section, removed old L1 Tags.

3D 64BPP and 128BPP special cases removed.

Change 91332 on 2003/03/20 by jhoule@jhoule_doc_lt

Minor correction to the Numbers table.
uINT gamma'd gave values from 0 to 15, but the 'range' entry was [-8..8) instead of [0, 16).
(reported by Daniel Willhite)
No version bump.

Change 91280 on 2003/03/20 by abeaudin@abeaudin_r400_win_marlboro

added gfx engine description

Change 91277 on 2003/03/20 by gregs@gregs_r400_win_marlboro

iodft insertion complete

Change 91262 on 2003/03/20 by dglen@dglen_r400

Updated with R500 line buffer size and core clock speed

Change 91227 on 2003/03/20 by mzhu@mzhu_crayola_win_tor

For MH_DC_LATENCY tests in chapter 3.3, force_read_delay_busy is forced to be less than or equal to force_read_delay_idle. This makes dmif model to send overlay data not later than graphics data for the first chunk of each line

Change 91209 on 2003/03/20 by jacarey@fl_jcarey_desktop

1. Updated bit width of Non-Prefetch counters in the CP_Non_Prefetch_Cntrs register in CP Spec.
2. Addition of INDIRECT_BUFFER_PFD packet to the PM4 Spec.

Change 91124 on 2003/03/19 by gregs@laptop1

differential pads dft + clean-up

Change 90952 on 2003/03/19 by sbagshaw@sbagshaw

DMIF tests clarified to use maximum pixel clock of 400 MHz. MH-DMIF latency tests modified so some use graphics & overlay surfaces whereas others only use a graphics surface
Details and procedure for CRTC interrupt test clarified.

Change 90883 on 2003/03/19 by mkelly@fl_mkelly_r400_win_laptop

SC debug register coverage...

Change 90874 on 2003/03/19 by abeaudin@abeaudin_r400_win_marlboro

more emulator information

Change 90871 on 2003/03/19 by abeaudin@abeaudin_r400_win_marlboro

remove doc

Change 90870 on 2003/03/19 by abeaudin@abeaudin_r400_win_marlboro

new emulator emulator information

Change 90869 on 2003/03/19 by abeaudin@abeaudin_r400_win_marlboro

remove doc

Change 90785 on 2003/03/18 by sbagshaw@sbagshaw

DMIF System and Stress tests (section 3.3 and 4.3) modified to utilize real display mode timings and clock speeds.

Change 90763 on 2003/03/18 by abeaudin@abeaudin_r400_win_marlboro

answers to software questions

Change 90744 on 2003/03/18 by mkelly@fl_mkelly_r400_win_laptop

Test para_enable bit, update register coverage...

Change 90739 on 2003/03/18 by jacarey@fl_jcarey_desktop

Add stall conditions for IB2D init w.r.t. in-flight indirect buffer inits.

Change 90715 on 2003/03/18 by vgoel@fl_vgoel2

added r400vgt_hos_pnt_adaptive_complex bug

Change 90699 on 2003/03/18 by gregs@gregs_r400_win_marlboro

DRAM_RST, TEST_YCLK, TEST_MCLK pads

Change 90670 on 2003/03/18 by mdoggett@MA_MDOGGETT_LT

some modifications towards version 0.5.
version 0.5 not yet completed.

Change 90659 on 2003/03/18 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 90613 on 2003/03/17 by csampayo@fl_csampayo_r400

Control flow, predicate and multi-context and multi-prim test. Upfdated test_list accordingly

Change 90606 on 2003/03/17 by gabarca@gabarca_crayola_win_cvd

added timin lines

Change 90592 on 2003/03/17 by moev@moev

updates to include dbist

Change 90492 on 2003/03/17 by khabbari@khabbari_r400_win

r400 last release

Change 90257 on 2003/03/14 by gregs@gregs_r400_win_marlboro

memory strobe signals' names back to MnWDQSn and MnRQDSn

Change 90251 on 2003/03/14 by mkelly@fl_mkelly_r400_win_laptop

sc_sp centers/centroids parameters 13, 14, 15

Change 90225 on 2003/03/14 by rthambim@rthambim_r400_win_tor

Added pci/agp address expansion, initiator register list.

Change 90161 on 2003/03/14 by mkelly@fl_mkelly_r400_win_laptop

sc_sp sample control parameters 8 - 13

Change 90160 on 2003/03/14 by jacarey@fl_jcarey_desktop

Another Stall Condition for Indirect_Buffer packet

Change 90158 on 2003/03/14 by jacarey@fl_jcarey_desktop

Updated Pseudocode for PREFETCH_DISABLE mode.

Change 90122 on 2003/03/13 by gregs@laptop1

update

Change 90107 on 2003/03/13 by georgev@devel_georgevh2_r400_win_marlboro

Fix _ to / typo.

Change 90102 on 2003/03/13 by georgev@devel_georgevh2_r400_win_marlboro

Updated sheets.

Change 90076 on 2003/03/13 by gregs@gregs_r400_win_marlboro

update

Change 90033 on 2003/03/13 by mkelly@fl_mkelly_r400_win_laptop

sc_sp sampling through interpolators, parameters 3 - 8

Change 90005 on 2003/03/13 by jacarey@fl_jcarey_desktop

Updates for Prefetch-Disable Mode to Fetching Indirect Buffers

Change 89973 on 2003/03/13 by georgev@devel_georgevh2_r400_win_marlboro

First Revision.

Change 89959 on 2003/03/13 by georgev@ma_georgev

Empty for first rev.

Change 89951 on 2003/03/13 by mkelly@fl_mkelly_r400_win_laptop

Prim type detection on gpr position 0, and 9 - 15

Change 89899 on 2003/03/12 by gregs@laptop1

parallel ROM supports 2 and 4Mbit parts.

Change 89876 on 2003/03/12 by jimmylau@jimmylau_r400_win_tor

Add details for the following R400 changes :

1. BIF coherency
2. 64-byte PCI and AGP writes
3. BIF performance counter
4. FW splitter that splits 128-bit data from FW to 64-bit data to HDP
5. Bug fix for AGP8x AD calibration

Change 89800 on 2003/03/12 by mkelly@fl_mkelly_r400_win_laptop

Primtype detection in the pixel shader, gpr positions 2 - 8

Change 89764 on 2003/03/12 by gregs@laptop1

update

Change 89759 on 2003/03/12 by georgev@devel_georgevh2_r400_win_marlboro

Added some more description.

Change 89756 on 2003/03/12 by mdoggett@MA_MDOGGETT_LT

Minor modifications

Change 89739 on 2003/03/12 by jacarey@fl_jcarey_desktop

Update to proposal.

Change 89737 on 2003/03/12 by gregs@laptop1

strap for "rom on vip" is 0001 now.

Change 89729 on 2003/03/12 by sbagshaw@sbagshaw

Sections 3.1 and 4.1 for DCCIF tests updated with new parameters from Mahendra's testing.

Change 89601 on 2003/03/11 by jacarey@fl_jcarey2

Proposal for Pre-Fetch Disabling

Change 89596 on 2003/03/11 by gregs@gregs_r400_win_marlboro

update

Change 89577 on 2003/03/11 by tien@ma_spinach

Added extra location for SAMPLE_LOCATION bit

Change 89569 on 2003/03/11 by mkelly@fl_mkelly_r400_win_laptop

Check POLY, POINT, LINE prim type detection in SP on parameter 0

Change 89556 on 2003/03/11 by gregs@gregs_r400_win_marlboro

update

Change 89544 on 2003/03/11 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 89501 on 2003/03/11 by jiezhou@cn_jiezhou

update from testplan review

Change 89429 on 2003/03/10 by csampayo@fl_csampayo_r400

Updated VGT tests status to 100% for all tests that pass hardware compare

Change 89424 on 2003/03/10 by csampayo@fl_csampayo_r400

Update VGT and SU sections

Change 89384 on 2003/03/10 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 89375 on 2003/03/10 by ygiang@ygiang_r400_win_marlboro_p4

updated: sp verification spreadsheet

Change 89331 on 2003/03/10 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 89313 on 2003/03/10 by csampayo@fl_csampayo_r400

Some housekeeping updates.

Change 89312 on 2003/03/10 by csampayo@fl_csampayo_r400

Housekeeping schedule

Change 89118 on 2003/03/07 by csampayo@fl_csampayo_r400

New test checking single/dual vertex vectors of various sizes. Updated test_list and test tracker accordingly

Change 89108 on 2003/03/07 by omesh@ma_omesh

Updated the spreadsheet to some extent, but still haven't finished....

Change 89075 on 2003/03/07 by fliljero@fl_frank

latest pass/fail results

Change 89066 on 2003/03/07 by moev@moev

r400 structure test

Change 89032 on 2003/03/07 by csampayo@fl_csampayo_r400

Added pass-thru test with large (>64 indices) vertex vectors. Updated test_list and test tracker accordingly

Change 89021 on 2003/03/07 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 88919 on 2003/03/06 by jiezhou@cn_jiezhou

update

Change 88913 on 2003/03/06 by csampayo@fl_csampayo_r400

Adding mixed VGT 2/1 output vectors tests

Change 88910 on 2003/03/06 by ashishs@fl_ashishs_r400_win

updated

Change 88905 on 2003/03/06 by gabarca@gabarca_crayola_win_cvd

updated reg table adding timing calcs

Change 88847 on 2003/03/06 by jacarey@fl_jcarey_desktop

Fix typo in pm4 spec.

Change 88832 on 2003/03/06 by gregs@gregs_r400_win_marlboro

update

Change 88766 on 2003/03/06 by ashishs@fl_ashishs_r400_win

updated for texture wrap bug which has been closed now

Change 88746 on 2003/03/06 by mkelly@fl_mkelly_r400_win_laptop

Completes initial check of prim type detection in the pixel shader
checking SQ POINT (r400sc_sp_sample_cntl_09), SQ LINE (r400sc_sp_sample_cntl_11)
and SQ POLY (this checkin).

Change 88591 on 2003/03/05 by mzhu@mzhu_crayola_win_tor

Change overlay base address to be different with graphics base address in chapter 3.4.8.8.

Change 88577 on 2003/03/05 by gregs@gregs_r400_win_marlboro

update

Change 88553 on 2003/03/05 by nbarbier@nbarbier_r400_win_tor

Updated section 9.4.

Change 88534 on 2003/03/05 by mkelly@fl_mkelly_r400_win_laptop

MultiPass Indirect Buffer multiple looping...

Change 88515 on 2003/03/05 by mkelly@fl_mkelly_r400_win_laptop

MultiPass Indirect Buffer / SC pixel LOOP interaction / 2 Segment / 2 Pass

Change 88445 on 2003/03/04 by csampayo@fl_csampayo_r400

More VGT pass-thru tests checking grouper data types

Change 88405 on 2003/03/04 by mzhu@mzhu_crayola_win_tor

Add BASE_ADDRESS = 384MB - 4KB case for graphics, overlay, icon and cursor in chapter 3.4.7.2, 3.4.7.3, 3.4.8.4 and 3.4.8.8.

Change 88370 on 2003/03/04 by gregs@gregs_r400_win_marlboro

update

Change 88360 on 2003/03/04 by grayc@grayc_r400_win

initial release of block validation blocks

Change 88347 on 2003/03/04 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 88327 on 2003/03/04 by fliljero@fl_frank

added wait_gfx_idle(); to force synchronization

Change 88312 on 2003/03/04 by fliljero@fl_frank

mem-mapped real-time SQ constant regs moved to a 16 word alignment - from 0x124a0 to 12500

Change 88220 on 2003/03/03 by jiezhou@cn_jiezhou

first testplan review

Change 88147 on 2003/03/03 by tien@ma_spinach

Let's try this again. same as prev rev.

Change 88146 on 2003/03/03 by tien@ma_spinach

Updated to include new bits in r400TxVtxInstConstSqTp.xls

Change 88116 on 2003/03/03 by mzhu@mzhu_crayola_win_tor

Change overlay gamma correction register default values to be linear.

Change 88023 on 2003/03/03 by ashishs@fl_ashishs_r400_win

updated CL special cases %

Change 88004 on 2003/03/03 by georgev@devel_georgevh2_r400_win_marlboro

Updated to reflect new tests.

Change 87906 on 2003/03/01 by gregs@gregs_r400_win_cc

CXTAL1SHV2, PGTMDSSHVA4

Change 87776 on 2003/02/28 by fliljero@fl_frank

update

Change 87764 on 2003/02/28 by smoss@smoss_crayola_win

SU tests

Change 87762 on 2003/02/28 by llefebvr@llefebvr_r400

Added the new SX interface.

Change 87752 on 2003/02/28 by frising@frising_r400_win_marlboro

v.1.68
-add CLAMP_DISABLE to vertex fetch constant.

Change 87636 on 2003/02/28 by gregs@gregs_r400_win_marlboro

update

Change 87501 on 2003/02/27 by mzhu@mzhu_crayola_win_tor

Update 5.9.10

Change 87476 on 2003/02/27 by fliljero@fl_frank

excel spreadsheet to track progress of test run the the gc testbench

Change 87468 on 2003/02/27 by gregs@gregs_r400_win_marlboro

update

Change 87452 on 2003/02/27 by mzhu@mzhu_crayola_win_tor

Add 5.9.9 More tests to improve code coverage
Add 5.9.10 Read F, V and H count when CRTC is disabled

Change 87298 on 2003/02/27 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 87190 on 2003/02/26 by gregs@laptop1

updated widths of new memory pads + initial distribution

Change 87189 on 2003/02/26 by gregs@laptop1

<updated board straps + added TVDAC placement>

Change 87175 on 2003/02/26 by csampayo@fl_csampayo_r400

Update test to use all 8 contexts, update test tracker accordingly

Change 87168 on 2003/02/26 by csampayo@fl_csampayo_r400

Add new VGT pass-thru tests

Change 87160 on 2003/02/26 by sbagshaw@sbagshaw

fixed some details of SCL_CP_coefficient_ram SCL system test

Change 87131 on 2003/02/26 by sbagshaw@sbagshaw

procedure and description for SCL_CP_coefficient_ram system test changed to correct methods.

Change 86921 on 2003/02/25 by gregs@gregs_r400_win_marlboro

ESD power and groud pads for memory interface

Change 86915 on 2003/02/25 by gregs@gregs_r400_win_marlboro

new crayola memory pads (breaks, corners, dcap, etc.)

Change 86894 on 2003/02/25 by csampayo@fl_csampayo_r400

Closing Bug# 72

Change 86772 on 2003/02/25 by jiezhou@cn_jiezhou

update

Change 86764 on 2003/02/25 by nbarbier@nbarbier_r400_win_tor

More updates to the Code Coverage Section.

Change 86735 on 2003/02/25 by hdong@hdong_r400_win-_tor

revert #11

Change 86713 on 2003/02/25 by ashishs@fl_ashishs_r400_win

updated for r400sq_16tex_interp_combo_01

Change 86709 on 2003/02/25 by jacarey@fl_jcarey_desktop

Initial Baseline

Change 86701 on 2003/02/25 by hdong@hdong_r400_win-_tor

delete disp1(2)_x_end, and disp1(2)_y_end.

Change 86691 on 2003/02/25 by donaldl@fl_donaldl_p4

SC clock diagram

Change 86673 on 2003/02/25 by scamlin@scamlin_crayola_win

sq clocks

Change 86661 on 2003/02/25 by mkelly@fl_mkelly_r400_win_laptop

Simple RTS for Christeen...

Change 86605 on 2003/02/24 by csampayo@fl_csampayo_r400

Adding pass-thru tests with 32 bit indices

Change 86600 on 2003/02/24 by nbarbier@nbarbier_r400_win_tor

More updates to Code Coverage Section.

Change 86575 on 2003/02/24 by jacarey@fl_jcarey2

Clarification of write confirm interval as experimental for R400
in the ME_INIT packet.

Change 86490 on 2003/02/24 by jennho@jennho_crayola0

New schematics for NPL.

Change 86489 on 2003/02/24 by jennho@jennho_crayola0

Updated schematics based on the lastest RTL changes.

Change 86488 on 2003/02/24 by gregs@gregs_r400_win_marlboro

update

Change 86444 on 2003/02/24 by nbarbier@nbarbier_r400_win_tor

Added section for additional tests required to satisfy code coverage (Section 9).

Change 86429 on 2003/02/24 by gabarca@gabarca_crayola_win_cvd

Fixed viewport x, y start becauase the surface also rotates

Change 86428 on 2003/02/24 by tshah@fl_tshah

Fixed wrong connection in VGT

Change 86415 on 2003/02/24 by jacarey@fl_jcarey_desktop

Comment to cp_int_cntl register

Change 86414 on 2003/02/24 by tshah@fl_tshah

Clock diagram for KS tile (RBBM+VGT+IDCT)

Change 86410 on 2003/02/24 by rramsey@RRAMSEY_P4_r400_win

Add clock gating diagram for sc_b

Change 86390 on 2003/02/24 by jacarey@fl_jcarey_desktop

Fix typo in indirect_buffer packet

Change 86374 on 2003/02/24 by jiezhou@cn_jiezhou

update

Change 86371 on 2003/02/24 by gabarca@gabarca_crayola_win_cvd

Clarified 7.3.2: we want underline and blink at the same time

Change 86350 on 2003/02/24 by mkelly@fl_mkelly_r400_win_laptop

Update SC status, re-assign most of the remaining cases to STRESS testing.

Change 86308 on 2003/02/23 by gregs@laptop1

added DRAM_SEL pins

Change 86288 on 2003/02/23 by gregs@gregs_r400_win_cc

new memory pads updated

Change 86181 on 2003/02/21 by gabarca@gabarca_crayola_win_cvd

after code coverage

Change 86144 on 2003/02/21 by nbarbier@nbarbier_r400_win_tor

Test Plan Update.

Change 86098 on 2003/02/21 by csampayo@fl_csampayo_r400

Some housekeeping

Change 86022 on 2003/02/21 by sbagshaw@sbagshaw

R400 DC test plan with updated numbers for CRTC timings for most modes to account for latency from Memory Hub (MH) to Display Composite Pipe (DCP).

Change 86005 on 2003/02/21 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 85948 on 2003/02/21 by mdoggett@MA_MDOGGETT_LT

Updated conditions for instruction and slice state transistions.

Change 85933 on 2003/02/21 by ashishs@fl_ashishs_r400_win

updated

Change 85893 on 2003/02/21 by mzhu@mzhu_crayola_win_tor

Update 3.4.9.13 and 3.4.9.14 to cover three cases in different display area, icon only, cursor only and icon is overlaped by cursor.

Change 85774 on 2003/02/20 by csampayo@fl_csampayo_r400

Adding point size clamping tests

Change 85692 on 2003/02/20 by mzhu@mzhu_crayola_win_tor

Set different surface address from 0 - 256MB for graphics, overlay, icon and cursor in chapter 3.4.7.2, 3.4.7.3, 3.4.8.4 and 3.4.8.8.

Change 85669 on 2003/02/20 by rfevreau@rfevreau_r400_win

Added 3 new tests for code coverage: 1) x2 on D2
2) x2 with cursor codes 1, 2, 3

Change 85602 on 2003/02/20 by mzhu@mzhu_crayola_win_tor

Update tests in 3.4.9.13 and 3.4.9.14

Change 85539 on 2003/02/20 by mdoggett@MA_MDOGGETT_LT

version 1.0 of out of order data return for TC. Will be eventually merged into TC spec.

Change 85504 on 2003/02/20 by gregs@gregs_r400_win_marlboro

update

Change 85486 on 2003/02/20 by gregs@gregs_r400_win_marlboro

update

Change 85475 on 2003/02/20 by mkelly@fl_mkelly_r400_win_laptop

RTS rectangle walk direction x_dir = 0, y_dir = 1

Change 85417 on 2003/02/19 by csampayo@fl_csampayo_r400

Some format cleanup

Change 85416 on 2003/02/19 by csampayo@fl_csampayo_r400

Some schedule update

Change 85411 on 2003/02/19 by smoss@smoss_crayola_win

update

Change 85403 on 2003/02/19 by gregs@gregs_r400_win_marlboro

minor update

Change 85397 on 2003/02/19 by ashishs@fl_ashishs_r400_win

updated

Change 85388 on 2003/02/19 by ashishs@fl_ashishs_r400_win

updated for the week of Feb 22nd

Change 85360 on 2003/02/19 by ashishs@fl_ashishs_r400_win

updated for 3 tests (need to be updated for 7 more)

Change 85345 on 2003/02/19 by ashishs@fl_ashishs_r400_win

updated

Change 85340 on 2003/02/19 by jiezhou@cn_jiezhou

Change FRC crtc from interlaces mode to progress mode

Change 85335 on 2003/02/19 by georgev@devel_georgevh2_r400_win_marlboro

Updated.

Change 85316 on 2003/02/19 by tshah@fl_tshah

clock diagram for the PD team

Change 85294 on 2003/02/19 by fliljero@fl_fliljeros

changed name of regclk_active signal coming from RBBM.
divided logic cloud into 2 separate clouds for the enables since they do not use the same logic to generate the enables.

Change 85269 on 2003/02/19 by frising@ma_frising

v.1.67
-FMT_10_11_11_AS_16_16_16_16 and FMT_11_11_10_AS_16_16_16_16 are not degammable.

Change 85204 on 2003/02/19 by jacarey@fl_jcarey_desktop

Fix typo for trans_bitblt for clr_cmp_src fields

Change 85203 on 2003/02/19 by csampayo@fl_csampayo_r400

More format updates

Change 85182 on 2003/02/19 by bhankins@fl_bhankins_r400_win

Initial checkin

Change 85164 on 2003/02/19 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 85140 on 2003/02/18 by csampayo@fl_csampayo_lt_r400

Added charts

Change 85126 on 2003/02/18 by gregs@laptop1

DRENB for dvo clock and control is high

Change 85118 on 2003/02/18 by jling@jling_crayola0

Added OE signal from Strobe to data nearpad in visio diagram

Change 85008 on 2003/02/18 by scamlin@scamlin_crayola_win

renamed and added stuff based on PD feedback

Change 84981 on 2003/02/18 by jiezhou@cn_jiezhou

updating test plan to reflect hardware changes.

Change 84967 on 2003/02/18 by gregs@laptop1

update

Change 84922 on 2003/02/18 by csampayo@fl_csampayo_r400

Initial checkin

Change 84920 on 2003/02/18 by mzhu@mzhu_crayola_win_tor

Correct register setting in 3.4.9.13 and 3.4.9.14

Change 84914 on 2003/02/18 by mzhu@mzhu_crayola_win_tor

Add 3.4.9.16 Data pattern for Graphics and Overlay Keyer Code Coverage

Change 84903 on 2003/02/18 by gregs@laptop1

improvements

Change 84807 on 2003/02/17 by mzhu@mzhu_crayola_win_tor

Add 3.4.9.13 Test YCbCr -> sRGB -> YCbCr
Add 3.4.9.14 Icon and Cursor Data Clamping
Add 3.4.9.15 Color Space Conversion Data Clamping

Change 84773 on 2003/02/17 by jiezhou@cn_jiezhou

updating for the second review

Change 84568 on 2003/02/14 by jhoule@MA_JHOULE

v 1.66:

Changed FMT_24_8* formats to only fetch the Z value, as this was the original scheme.
Stencil reads can be done by using FMT_8_8_8_8 using another constant.

Updated degamma comments to account for the fact that NUM_FORMAT_ALL must be set to RF, since degamma only makes sense on uRF source.

Change 84541 on 2003/02/14 by jling@jling_crayola0

Removed VMODE0/1 from memory section

Change 84535 on 2003/02/14 by vgoel@fl_vgoel2

updated register coverage

Change 84527 on 2003/02/14 by jling@jling_crayola0

R400 io pad drawing (premlinimary)

Change 84505 on 2003/02/14 by mkelly@fl_mkelly_r400_win_laptop

Real Time Stream Line List Primitive

Change 84420 on 2003/02/14 by csampayo@fl_csampayo_r400

Added multi context tests

Change 84336 on 2003/02/13 by jowang@jowang_R400_win

updated after code coverage review

Change 84290 on 2003/02/13 by mkelly@fl_mkelly_r400_win_laptop

RTS intertwined with Hi-Z viz query

Change 84278 on 2003/02/13 by georgev@devel_georgevh2_r400_win_marlboro

Updated with new tests.

Change 84270 on 2003/02/13 by gabarca@gabarca_crayola_win_cvd

Fixed

Change 84264 on 2003/02/13 by alleng@alleng_r400_win_marlboro

Initial submission of R400 notes

Change 84235 on 2003/02/13 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 84122 on 2003/02/13 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 83959 on 2003/02/12 by ygiang@ygiang_r400_win_marlboro_p4

updated: cp cntrl reg for perf counters

Change 83839 on 2003/02/12 by mdoggett@mdoggett_r400_win_platypus

Changed all Two Layer L2 Block Offsets.
Changed SM3.
Added formats 54,55,56 changed 7,16, 17.
Updated top level to reflect block split.

Change 83798 on 2003/02/12 by mdoggett@MA_MDOGGETT_LT

Changed all Two Layer L2 Block Offsets.
Changed SM3.
Added formats 54,55,56 changed 7,16, 17.
Updated top level to reflect block split.

Change 83705 on 2003/02/11 by jiezhou@cn_jiezhou

updating from the design review

Change 83646 on 2003/02/11 by jiezhou@cn_jiezhou

update

Change 83605 on 2003/02/11 by ashishs@fl_ashishs_r400_win

updated

Change 83597 on 2003/02/11 by jhoule@MA_JHOULE

Final description of mipfilter point (potential reuse of trijuice)

Change 83555 on 2003/02/11 by jiezhou@cn_jiezhou

clock diagram for design review

Change 83549 on 2003/02/11 by jacarey@fl_jcarey_desktop

Allow SRC_H/W != DST_H/W for all AlphaBlend OPs in Microcode
Updated associated documents.

Change 83530 on 2003/02/11 by jhoule@MA_JHOULE

0.9.20
Added mip rounding in LOD computation pseudo-code

Change 83515 on 2003/02/11 by jiezhou@cn_jiezhou

Increase H-total in DMIF stress tests.

Change 83493 on 2003/02/11 by jimmylau@jimmylau_r400_win_tor

Update the table for AP_SIZE in section 6.1 because AP_SIZE is the same for a particular aperture size strap, regardless of multifunction.

Change 83389 on 2003/02/10 by nbarbier@nbarbier_r400_win_tor

Updated document with list of signals in DC/IO interface that don't go through DCIO block.

Change 83328 on 2003/02/10 by jhoule@MA_JHOULE

Updated replication table

Change 83253 on 2003/02/10 by csampayo@fl_csampayo_r400

Housekeeping SU section

Change 83238 on 2003/02/10 by csampayo@fl_csampayo_r400

Update real time parameter registers

Change 83233 on 2003/02/10 by mkelly@fl_mkelly_r400_win_laptop

Use all 4 SQ parameters for RT streams...

Change 83130 on 2003/02/09 by gregs@laptop1

update for netlist rev4.

Change 83084 on 2003/02/08 by jimmylau@jimmylau_r400_win_tor

Modify diagram on section 6.1 to illustrate change of HDP/VGA/RBBM interface
Add tables in section 6.1 to illustrate the new strap settings for memory/register/ROM aperture size
Add description to the PCI spec. Rev2.3 support

Change 83080 on 2003/02/08 by gregs@laptop1

MC_IO_wr_strb + DRAM_SEL + VREFs

Change 82947 on 2003/02/07 by csampayo@fl_csampayo_r400

Updates to SQ_PROGRAM_CNTL

Change 82944 on 2003/02/07 by jacarey@fl_jcarey2

Update Spec

Change 82939 on 2003/02/07 by jowang@jowang_R400_win

added Early1 and Early2 states

Change 82927 on 2003/02/07 by nbarbier@nbarbier_r400_win_tor

Minor changes.

Change 82845 on 2003/02/07 by csampayo@fl_csampayo_r400

Add missing sq_basic_test test case

Change 82838 on 2003/02/07 by llefebvr@llefebvr_r400

small update regarding the implementation change for the Pos Allocated / PC allocated bits.

Change 82808 on 2003/02/07 by mkelly@fl_mkelly_r400_win_laptop

Validate dummy quad deallocation in pixel vector buffer is good.

Change 82734 on 2003/02/06 by gregs@laptop1

added VREFs

Change 82699 on 2003/02/06 by csampayo@fl_csampayo_r400

Remove non-existing tests (randomized)

Change 82684 on 2003/02/06 by nbarbier@nbarbier_r400_win_tor

Updated Section 8.

Change 82683 on 2003/02/06 by csampayo@fl_csampayo_r400

Adding Marl test list tracker

Change 82639 on 2003/02/06 by georgev@devel_georgevh2_r400_win_marlboro

Really SQ, that's why it's deleted.

Change 82560 on 2003/02/06 by ashishs@fl_ashishs_r400_win

updated to CL/VTE to 99% . Changed the %formula to account for the CL_POINT_SIZE register not being used.

Change 82554 on 2003/02/06 by rfevreau@rfevreau_r400_win

First test using ico file for data

Change 82543 on 2003/02/06 by dwong@cndwong2

added details on reset schemes

Change 82508 on 2003/02/06 by jacarey@fl_jcarey_desktop

Update NQ Flag for Micro Engine's DMA Engine

Change 82497 on 2003/02/06 by gregs@gregs_r400_win_marlboro

update

Change 82489 on 2003/02/06 by ashishs@fl_ashishs_r400_win

updated

Change 82429 on 2003/02/06 by mkelly@fl_mkelly_r400_win_laptop

Simplified version of r400sc_rts_12 for regress_e

Change 82426 on 2003/02/06 by ashishs@fl_ashishs_r400_win

updated

Change 82278 on 2003/02/05 by csampayo@fl_csampayo_r400

Update to better see register coverage

Change 82256 on 2003/02/05 by csampayo@fl_csampayo_r400

Add field to PA_CL_ENHANCE

Change 82239 on 2003/02/05 by ashishs@fl_ashishs_r400_win

updated

Change 82235 on 2003/02/05 by gregs@gregs_r400_win_marlboro

netlist revision 4

Change 82213 on 2003/02/05 by jacarey@fl_jcarey_desktop

Correct miscellaneous typo's in the documetn

Change 82184 on 2003/02/05 by csampayo@fl_csampayo_r400

Increase max memory size for tests. Update spreadsheet

Change 82156 on 2003/02/05 by georgev@ma_georgev

Renamed file.

Change 82059 on 2003/02/05 by gregs@gregs_r400_win_marlboro

revision 4 netlist release

Change 81881 on 2003/02/04 by llefebvr@llefebvr_r400

added some more comments in the spreadsheet.

Change 81860 on 2003/02/04 by jacarey@fl_jcarey_desktop

Miscellaneous Documentation Updates

Change 81791 on 2003/02/04 by mkelly@fl_mkelly_r400_win_laptop

RTS intertwined with Viz Query and kill_pix_post_detail_mask.

Change 81731 on 2003/02/04 by mkelly@fl_mkelly_r400_win_laptop

Viz Query intertwined with RT streams, complete...

Change 81712 on 2003/02/04 by jacarey@fl_jcarey_desktop

Update write-only status of microcode read address registers.

Change 81667 on 2003/02/03 by gregs@laptop1

fixed bug introduced earlier

Change 81620 on 2003/02/03 by gregs@gregs_r400_win_marlboro

cleaned JTAG interface

Change 81538 on 2003/02/03 by llefebvr@llefebvr_r400

Missed bits 41 and 42 in the SQ_EXEC instruction format. Those are RESERVED as well.

Change 81486 on 2003/02/03 by ygiang@ygiang_r400_win_marlboro_p4

added: sp coverage

Change 81412 on 2003/02/03 by ashishs@fl_ashishs_r400_win

updated

Change 81401 on 2003/02/03 by llefebvr@llefebvr_r400

refined the interfaces to the SP to specify wich signals should or shouldn't be pipelined.

Change 81382 on 2003/02/03 by jacarey@fl_jcarey_desktop

Add clock gating diagram to spec.
Add note that debug data I/O is asynchronous.

Change 81347 on 2003/02/02 by gregs@laptop1

update

Change 81312 on 2003/02/02 by gregs@gregs_r400_win_cc

update

Change 81261 on 2003/02/01 by gregs@gregs_r400_win_cc

added note that debug bus is asynchronous and there shopuld be no registers on inputs or outputs.

Change 81219 on 2003/01/31 by lchen@lchen_crayola0

MEM IO schematics and netlist

Change 81215 on 2003/01/31 by lchen@lchen_crayola0

version 4

Change 81159 on 2003/01/31 by vgoel@fl_vgoel2

updated VGT register coverage from TE setup

Change 81126 on 2003/01/31 by fliljero@fl_frank

new drawing for PD team

Change 81080 on 2003/01/31 by jacarey@fl_jcarey_desktop

Fix Typo in ME_INIT packet.

Change 81008 on 2003/01/31 by jacarey@fl_jcarey_desktop

1. Correct Width of Microcode RAM read and write registers.
2. Fix re-ordering queue data available determination
3. Document updates for #1.

Change 80832 on 2003/01/30 by llefebvr@llefebvre_laptop_r400

wording change for the predicate override bit.

Change 80785 on 2003/01/30 by jacarey@fl_jcarey_desktop

Clarify that ME_INIT invalidates pointers only if processed in a non-real-time stream.
This does not happen if it is processed in a real-time stream.

Change 80783 on 2003/01/30 by georgev@ma_georgev

First part of register list for tests. Not finished due to pending changes.

Change 80707 on 2003/01/30 by mkelly@fl_mkelly_r400_win_laptop

Update comments, RTS with SC quad order enable toggling...

Change 80686 on 2003/01/30 by mkelly@fl_mkelly_r400_win_laptop

RTS and SC FIFO sizing combinations...

Change 80682 on 2003/01/30 by jacarey@fl_jcarey_desktop

Pseudocode Update #2 for today

Change 80680 on 2003/01/30 by jacarey@fl_jcarey_desktop

Update ME_INIT in Pseudocode Land

Change 80679 on 2003/01/30 by mkelly@fl_mkelly_r400_win_laptop

RTS combinations with Vtx and Pix pipes 0/2 disabled with
SC one quad per clock toggled, shader back pressure,
interpolator shading toggling

Change 80664 on 2003/01/30 by jacarey@fl_jcarey_desktop

1. Reserved bits in CP_DEBUG register are preserved.
2. Added default for CP_INT_STAT register.
3. Add number_dword=0 check for PolyScanLines and HostData_Blt packets.
4. Associated documentation updates for above items.
5. Update full chip tests for above items.

Change 80503 on 2003/01/29 by csampayo@fl_csampayo_r400

Updated VGT section

Change 80479 on 2003/01/29 by mkelly@fl_mkelly_r400_win_laptop

Vtx and pix pipes 2 and 3 disabled with RTS triangles and rectangles and non-RTS stipple lines, complete

Change 80426 on 2003/01/29 by rfevreau@rfevreau_r400_win

Changed tests to use bitmaps for cursor data

Change 80422 on 2003/01/29 by jacarey@fl_jcarey_desktop

Add note to spec regarding the preservation of "reserved" bits in the cp_debug register.

Change 80420 on 2003/01/29 by jennho@jennho_crayola0

Added ADDR/Command/RD_DATA/DIM signals floorplan.

Change 80401 on 2003/01/29 by csampayo@fl_csampayo_r400

Update VGT section

Change 80398 on 2003/01/29 by csampayo@fl_csampayo_r400

Adding new VGT test for missing reg coverage

Change 80305 on 2003/01/29 by mkelly@fl_mkelly_r400_win_laptop

Check stippled line integrity with real time streams, complete.

Change 80022 on 2003/01/28 by nkociuk@ma_nkociuk

update TP perfcounter events

Change 79996 on 2003/01/28 by gregs@gregs_r400_win_marlboro

first release of the new memory pads.

Change 79984 on 2003/01/28 by mkelly@fl_mkelly_r400_win_laptop

Polymode RTS test

Change 79957 on 2003/01/28 by kcorrell@kcorrell_r400_docs_marlboro_nb

edited hi interface description

Change 79899 on 2003/01/28 by hartogs@fl_hartogs

Updated VGT_SQ interface descripttion.

Change 79824 on 2003/01/27 by csampayo@fl_csampayo_r400

Updated VGT status

Change 79818 on 2003/01/27 by csampayo@fl_csampayo_r400

Adding new VGT fifo tests

Change 79762 on 2003/01/27 by tshah@fl_tshah

added RBBM_CGM_soft_reset (hardware+emulator+tests+doc)

Change 79738 on 2003/01/27 by llefebvr@llefebvre_laptop_r400

some updates.

Change 79725 on 2003/01/27 by gregs@gregs_r400_win_marlboro

mem interface tests PASSED.

Change 79674 on 2003/01/27 by georgev@ma_georgev

First revision.

Change 79649 on 2003/01/27 by lchen@lchen_crayola0

checked in the shivah NPL schematics

Change 79613 on 2003/01/27 by ashishs@fl_ashishs_r400_win

added test description and updated tracker

Change 79549 on 2003/01/26 by gregs@laptop1

update

Change 79547 on 2003/01/26 by gregs@laptop1

update

Change 79527 on 2003/01/26 by gregs@laptop1

mem interface clean (except address from MCs)

Change 79368 on 2003/01/24 by gregs@gregs_r400_win_marlboro

update

Change 79284 on 2003/01/24 by georgev@ma_georgev

Added a few tests. Yung needs file for his nefarious purposes.

Change 79270 on 2003/01/24 by jimmylau@jimmylau_r400_win_tor

First draft of BIF coherency test cases for R400

Change 79256 on 2003/01/24 by ashishs@fl_ashishs_r400_win

updated

Change 79235 on 2003/01/24 by mkelly@fl_mkelly_r400_win_laptop

RT provoking vertex looking good through interpolator on one parameter.

Change 79191 on 2003/01/24 by efong@efong_r400_win_tor_doc

moved gord to ltis188 and syang to ltis186

Change 79181 on 2003/01/24 by csampayo@fl_csampayo_r400

Updated VGT section

Change 79176 on 2003/01/24 by jhoule@MA_JHOULE

Changed DXT from 5/6 to 8 always.
Only kept first 2 sheets (should be enough).

Change 79174 on 2003/01/24 by mkelly@fl_mkelly_r400_win_laptop

Add modified and shortened version of r400sc_rts_09 (back face check on nonRT vs RT prims) to regress_e

Change 79161 on 2003/01/24 by mkelly@fl_mkelly_r400_win_laptop

Final, validating Pixel Shader face bit detection from sc_sp with nonRT and RT primtives

Change 79155 on 2003/01/24 by jacarey@fl_jcarey_desktop

Clarify update of constant write enables for LCC packet.

Change 79094 on 2003/01/23 by ashishs@fl_ashishs_r400_win

updated to remove the clip error detect 0%

Change 79063 on 2003/01/23 by csampayo@fl_csampayo_r400

Initial checkin

Change 78984 on 2003/01/23 by llefebvr@llefebvr_r400

small correction on memory export buffer sizes.

Change 78972 on 2003/01/23 by khabbari@khabbari_r400_win

changed the syncgen test list

Change 78962 on 2003/01/23 by mzhu@mzhu_crayola_win_tor

Update for MH-DC latency tests in chapter 3.3

Change 78947 on 2003/01/23 by jimmylau@jimmylau_r400_win_tor

Update after review on Jan 23, 03

Change 78917 on 2003/01/23 by jacarey@fl_jcarey_desktop

Fix LCC and Set Constant for incremental register updates.

Change 78913 on 2003/01/23 by efong@efong_r400_win_tor_doc

Added in Project linux assignments excel spreadsheet

Change 78876 on 2003/01/23 by mkelly@fl_mkelly_r400_win_laptop

Validate face bit in pixel shader for multi-tile coverage prims

Change 78874 on 2003/01/23 by ashishs@fl_ashishs_r400_win

updated to have percentages for the CL/VTE registers with comments

Change 78771 on 2003/01/22 by jennho@jennho_crayola0

Preliminary r400 MEM IO ring visio diagram.

Change 78733 on 2003/01/22 by ashishs@fl_ashishs_r400_win

updated

Change 78717 on 2003/01/22 by jimmylau@jimmylau_r400_win_tor

Initial Action items for BIF

Change 78693 on 2003/01/22 by gregs@gregs_r400_win_cc

update

Change 78663 on 2003/01/22 by gregs@gregs_r400_win_cc

update

Change 78658 on 2003/01/22 by jacarey@fl_jcarey_desktop

Update for AlphaBlend for ARGB1555 and Alpha_Source Blending

Change 78557 on 2003/01/22 by jhoule@MA_JHOULE

Changed *_FLOAT formats to have fast path available under VFetch only.
This means TFetches only do 32b/clock.

Corrected cycle multiplier for FMT_32_32_32_32 (was set to x3 instead of x4).

Added REQUEST_LATENCY field to constants. Controls out-of-order behavior.

Change 78432 on 2003/01/22 by jacarey@fl_jcarey_desktop

Fix Typo in Boolean Descriptions

Change 78374 on 2003/01/21 by lkang@lkang_r400_win_tor

update

Change 78369 on 2003/01/21 by gregs@gregs_r400_win_cc

Tuesday

Change 78345 on 2003/01/21 by ashishs@fl_ashishs_r400_win

updated

Change 78304 on 2003/01/21 by frising@ma_frising

v.1.93
-Z export from pixel sahder now in X channel.
-updated mova, kill and predicate instructions coissue rules now that we have a separate bus for mova results.
-add note saying input modifiers do not apply to PreviousScalar.
-show not used opcodes.
-add clamping to mova result sent to SQ.
-add 6 new scalar instructions that operate on a constant and GPR and associated documentation.

Change 78286 on 2003/01/21 by grayc@grayc_r400_win

connections of gfx pipeline

Change 78267 on 2003/01/21 by jennho@jennho_crayola0

Preliminary NPL schematics; not simulated yet.

Change 78098 on 2003/01/21 by gregs@gregs_r400_win_cc

Monday update

Change 78045 on 2003/01/20 by alleng@alleng_r400_win_marlboro

Added updated information regarding the RB performance counters...

Change 77987 on 2003/01/20 by csampayo@fl_csampayo_r400

Initial checkin

Change 77984 on 2003/01/20 by mzhu@mzhu_crayola_win_tor

Update DMIF model force signal names for MH-DC latency tests in chapter 3.3

Change 77922 on 2003/01/20 by ashishs@fl_ashishs_r400_win

updated

Change 77916 on 2003/01/20 by jacarey@fl_jcarey_desktop

1. Pixel Shader and Microcode to Set B6 for AAFONT packets.
2. Vertex Shader to ignore B10 for packets with Embedded Source.
3. Updated RB_BlendControl Settings for AlphaBlend Packet.
4. Associated Documentation for above changes.

Change 77910 on 2003/01/20 by mzhu@mzhu_crayola_win_tor

Add MH-DC latency tests in chapter 3.3

Change 77887 on 2003/01/20 by csampayo@fl_csampayo_r400

Some CL housekeeping

Change 77866 on 2003/01/20 by ashishs@fl_ashishs_r400_win

updated

Change 77848 on 2003/01/20 by jacarey@fl_jcarey_desktop

Note to Ply_NextScan and NextChar packets about required preceeding packets.

Change 77665 on 2003/01/17 by jennho@jennho_crayola0

<Preliminary R400 MEM IO timing spreadsheet.>

Change 77661 on 2003/01/17 by jennho@jennho_crayola0

<Preliminary R400 MEM IO timing spreadsheet.>

Change 77619 on 2003/01/17 by gregs@gregs_r400_win_marlboro

Friday update

Change 77558 on 2003/01/17 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 77554 on 2003/01/17 by mkelly@fl_mkelly_r400_win_laptop

Rectangle and triangle real time stream initial functional

Change 77520 on 2003/01/17 by jacarey@fl_jcarey2

Miscellaneous Comments to Registers

Change 77493 on 2003/01/17 by georgev@ma_georgev

Added descriptions of SQ_TESTS.

Change 77490 on 2003/01/17 by ashishs@fl_ashishs_r400_win

updated

Change 77459 on 2003/01/17 by lchen@lchen_crayola0

initial release of the MEM IO schematics and spice netlist

Change 77457 on 2003/01/17 by jacarey@fl_jcarey_desktop

1. Updated RB_BlendControl for AlphaBlend Packet
2. Added Microcode for Source Rotation for 2D.
3. Updated Documentation for Source Rotation and RB_BlendControl

Change 77306 on 2003/01/16 by gregs@gregs_r400_win_marlboro

fixed AGP clock layer (damaged by accident ..)

Change 77297 on 2003/01/16 by gregs@gregs_r400_win_marlboro

Thursday update

Change 77273 on 2003/01/16 by ashishs@fl_ashishs_r400_win

updated

Change 77174 on 2003/01/16 by georgev@ma_georgev

Added new list of SQ tests for Florida.

Change 77167 on 2003/01/16 by jennho@jennho_crayola0

<rv350 mem IO timing spreadsheet>

Change 77102 on 2003/01/16 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 77097 on 2003/01/16 by kcorrell@kcorrell_r400_docs_marlboro_nb

Fixed error in mhr_mhs_word, mhb_mhs_word (made description match hardware).
Updated update path to read cache from merge logic.

Change 77093 on 2003/01/16 by llefebvr@llefebvr_r400

Modified the alloc instruction to include a no-serial bit.

Change 77086 on 2003/01/16 by mkelly@fl_mkelly_r400_win_laptop

12 non real time packets of one triangle, each with 16 real time rectangle streams...

Change 77014 on 2003/01/15 by gregs@laptop1

updated address and CKE pads.

Change 77001 on 2003/01/15 by llefebvr@llefebvr_r400

Interface change from the SX (alloc dealloc bus) and interface change from the SP (predicates and kill mask).

Change 76917 on 2003/01/15 by mzhu@mzhu_crayola_win_tor

Add test 3 for full width icon and cursor at 3.4.9.12

Change 76894 on 2003/01/15 by ashishs@fl_ashishs_r400_win

removed vap_vte_vec0_13 since redundant with vap_vte_vec0_05

Change 76893 on 2003/01/15 by ashishs@fl_ashishs_r400_win

updated

Change 76880 on 2003/01/15 by vgoel@fl_vgoel2

updated bug report to date

Change 76868 on 2003/01/15 by lseiler@lseiler_r400_win_marlboro

Fixed a bug in the 3D tiling equation

Change 76849 on 2003/01/15 by gregs@laptop1

updated MCLK templates, macros, connections.

Change 76833 on 2003/01/15 by frising@ma_frising

v.1.64
-no functional changes just clean-up.
--make description of unused field in texture fetch instruction state that it is unused.
--add a missing outline border to above unused field.
--merge comments 14.) and 16.) in R400_DATA_FORMAT table section.
--make texture constant fields fit on to two pages.

Change 76816 on 2003/01/15 by ashishs@fl_ashishs_r400_win

updated

Change 76812 on 2003/01/15 by ashishs@fl_ashishs_r400_win

updated

Change 76807 on 2003/01/15 by ashishs@fl_ashishs_r400_win

removed r400vte_pos_neg_combo_04 since redundant with r400vte_pos_neg_combo_01

Change 76746 on 2003/01/14 by gregs@laptop1

updated DIM pads and macro.

Change 76730 on 2003/01/14 by gregs@laptop1

Tuesday (01-14-03) work in progress

Change 76723 on 2003/01/14 by ashishs@fl_ashishs_r400_win

updated

Change 76708 on 2003/01/14 by jhoule@MA_JHOULE

1.63
FMT_2_10_10_10 was wrongly set as degamma'able.
The new FMT_2_10_10_10_AS_16_16_16_16 must be used instead.

Change 76701 on 2003/01/14 by jiezhou@cn_jiezhou

updating frc settings

Change 76644 on 2003/01/14 by csampayo@fl_csampayo_r400

Updated HOS status

Change 76582 on 2003/01/14 by mkelly@fl_mkelly_r400_win_laptop

Update plan chart...

Change 76578 on 2003/01/14 by mkelly@fl_mkelly_r400_win_laptop

Update SC...

Change 76576 on 2003/01/14 by kcorrell@kcorrell_r400_docs_marlboro_nb

Added position of 3 bit endian field in Tag Buffer Contents table.
Changed mhr_mhs interface definition to support 128 bit transfers.
Fixed a couple of typo's.

Change 76557 on 2003/01/14 by mkelly@fl_mkelly_r400_win_laptop

OGL Rasterization validation...

Change 76417 on 2003/01/14 by mkelly@fl_mkelly_r400_win_laptop

Simple test, validate DX rasterization rules...

Change 76391 on 2003/01/13 by grayc@grayc_r400_win

updated for read test

Change 76388 on 2003/01/13 by gregs@laptop1

work in progress on memory interface.
Monday 01-13-03.
data and strobe templates + data PadList

Change 76366 on 2003/01/13 by ashishs@fl_ashishs_r400_win

CL/VTE final synch complete

Change 76364 on 2003/01/13 by ashishs@fl_ashishs_r400_win

added 3 tests (2 barycentric and 1 DX/OGL space test to individual submission count)

Change 76361 on 2003/01/13 by ashishs@fl_ashishs_r400_win

CL/VTE are perfectly updated after this submission

Change 76311 on 2003/01/13 by ashishs@fl_ashishs_r400_win

updated

Change 76298 on 2003/01/13 by jiezhou@cn_jiezhou

update dto increment (13bits frac represents 20bits frac)

Change 76290 on 2003/01/13 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 76278 on 2003/01/13 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 76269 on 2003/01/13 by ashishs@fl_ashishs_r400_win

updated for r400vte_coverage_02(removed)

Change 76263 on 2003/01/13 by ashishs@fl_ashishs_r400_win

updated for r400vte_coverage_02 since that test is deleted. Just decreased the count from the VT coverage.

Change 76199 on 2003/01/13 by jacarey@fl_jcarey_desktop

Documentation for the 2D Endian Mode Programming

Change 76196 on 2003/01/13 by jhoule@MA_JHOULE

1.62:
- Added formats FMT_2_10_10_10_AS_16_16_16_16, FMT_10_11_11_AS_16_16_16_16, and FMT_11_11_10_AS_16_16_16_16, which is equivalent to the old TFetch values.
- Changed FMT_2_10_10_10, FMT_10_11_11, and FMT_11_11_10 to be fast and unfilterable (equivalent
to the old VFetch).
- Added Filterable? column.
- Added comments regarding undefined behavior when filter is not set to Point.

Change 76174 on 2003/01/13 by jhoule@MA_JHOULE

1.61:
Single LOD_BIAS, located where LOD_BIAS_H was.
Added GRAD_EXP_ADJUST_{H|V} which should achieve the intended functionality of the dual-bias scheme (non-square-pixel resolve, and scanline-interleaved rendering under multi-chip).

Corrected minor typo (performace -> performance) on Features page.

Change 76169 on 2003/01/13 by csampayo@fl_csampayo_r400

Revised plan for the VGT and SU and updated status for the following tests:
r400vgt_dma_index_primtypes_02
r400vgt_real_time_events_07

Change 76117 on 2003/01/13 by ashishs@fl_ashishs_r400_win

updated

Change 76067 on 2003/01/12 by gregs@laptop1

update - pad ring - work in progress

Change 76066 on 2003/01/12 by jiezhou@cn_jiezhou

Initial release

Change 75932 on 2003/01/10 by gregs@gregs_r400_win_marlboro

update

Change 75926 on 2003/01/10 by rherrick@ma_rherrick_crayola

Implemented DC Urgent latency checker... Also implemented parser hooks to support Bandwidth checking parameters.

Change 75914 on 2003/01/10 by jiezhou@cn_jiezhou

make tests smaller

Change 75815 on 2003/01/10 by jacarey@fl_jcarey_desktop

Update equation used by micro engine for 1D sources for the SRC_X / Y terms.

Change 75811 on 2003/01/10 by rfevreau@rfevreau_r400_win

Cursor test fixes

Change 75642 on 2003/01/09 by csampayo@fl_csampayo_r400

Added 1 VGT performance and 1 VGT debug case, updated test_list and tracker accordingly

Change 75512 on 2003/01/09 by jacarey@fl_jcarey_desktop

1. Add level of indirection to the And_Mask and Or_Mask in the Reg_RMW packet.
2. Updated Associated Unit Test.
3. Updated PM4 Spec Accordingly.
4. Added Note to Polyline packet, that the scan_count needs to be 1 or greater.

Change 75356 on 2003/01/08 by hartogs@fl_hartogs

See version update info in document.

Change 75303 on 2003/01/08 by mkelly@fl_mkelly_r400_win_laptop

Stress vtx and pix pipe disable combinations with stippled LINE_LIST

Change 75236 on 2003/01/08 by ashishs@fl_ashishs_r400_win

updated for clip disable count

Change 75232 on 2003/01/08 by ashishs@fl_ashishs_r400_win

updated for new tests. Also reduced the count for clip disable tests from 4 to 1

Change 75199 on 2003/01/08 by ashishs@fl_ashishs_r400_win

updated(removed r400cl_ucp_pointlist_01)

Change 75197 on 2003/01/08 by jacarey@fl_jcarey_desktop

1. Reduction of queue sizes in CP for area savings.

Change 75191 on 2003/01/08 by jhoule@MA_JHOULE

Added TC/TP replication table and associated explanation.
Added a few tables to table of content.

Change 75183 on 2003/01/08 by vromaker@MA_VIC_P4

latest version of sq top level block diagram

Change 75173 on 2003/01/08 by csampayo@fl_csampayo_r400

Added new VGT pass-thru block test, updated test_list and test tracker accordingly.

Change 75002 on 2003/01/07 by gregs@gregs_r400_win_marlboro

DDC1data, DDC1clk, VSYNCA, HSYNCA, VSYNCB, and HSYNCB now incluided in DFT chain.

Change 74942 on 2003/01/07 by llefebvr@llefebvre_laptop_r400

New revision of the spec.

Change 74928 on 2003/01/07 by jiezhou@cn_jiezhou

update FRC DTO_INC parameters

Change 74922 on 2003/01/07 by jiezhou@cn_jiezhou

See the reversion description

Change 74919 on 2003/01/07 by mdoggett@mdoggett_r400_win_platypus

Fixed figure for TCA

Change 74887 on 2003/01/07 by gregs@gregs_r400_win_marlboro

update

Change 74783 on 2003/01/06 by gregs@gregs_r400_win_marlboro

added bad pipe disable register drawing.

Change 74742 on 2003/01/06 by ashishs@fl_ashishs_r400_win

updated comments for r400cl_ucp_cube_01

Change 74715 on 2003/01/06 by jacarey@fl_jcarey_desktop

Clarification of Invalidate_State packet w.r.t. use of Mem_Write

Change 74689 on 2003/01/06 by ashishs@fl_ashishs_r400_win

updated

Change 74686 on 2003/01/06 by ashishs@fl_ashishs_r400_win

updated

Change 74684 on 2003/01/06 by jacarey@fl_jcarey_desktop

Update width of brush offset to reflect 24 bits supported by the VGT.

Change 74668 on 2003/01/06 by vgoel@fl_vgoel2

added bug 1040

Change 74619 on 2003/01/06 by sbagshaw@sbagshaw

completed subsections of section 7 regarding the DAC output interface and DAC automatic device detection circuit

Change 74589 on 2003/01/06 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 74510 on 2003/01/04 by gregs@laptop1

added BAD_PIPE_DISBLE_REGISTER logic diagram

Change 74357 on 2003/01/03 by mkelly@fl_mkelly_r400_win_laptop

256 points per packet, all 15 legal combinations of vtx pipe disable

Change 74351 on 2003/01/03 by rfevreau@rfevreau_r400_win

New goldens, put rg files back to 31 tests

Change 74331 on 2003/01/03 by csampayo@fl_csampayo_r400

Closed bugs# 102 and 103

Change 74279 on 2003/01/03 by smoss@smoss_crayola_win

gave credit to vivian for her three read tests

Change 74201 on 2003/01/02 by jiezhou@cn_jiezhou

update for frame rate conversion SCL_VIDCAP_frame_rate_conv

Change 74196 on 2003/01/02 by frising@ma_frising

v.0.99e

-MUL_PREV2 scalar instruction now checks if PreviousScalar or input1.x is a NaN and returns -MAX_FLOAT if so.

Change 74194 on 2003/01/02 by frising@ma_frising

v.1.92

-MUL_PREV2 scalar instruction now checks if PreviousScalar or SrcC.X is a NaN and returns -MAX_FLOAT if so.

Change 74184 on 2003/01/02 by csampayo@fl_csampayo_lt_r400

Updated test and test_list for test r400vgt_real_time_events_06 and updated description/status on the test tracker for tests:
r400vgt_real_time_events_04
r400vgt_real_time_events_05
r400vgt_real_time_events_06

Change 74154 on 2003/01/02 by gregs@gregs_r400_win_marlboro

update

Change 74152 on 2003/01/02 by gregs@gregs_r400_win_marlboro

update

Change 74142 on 2003/01/02 by gregs@gregs_r400_win_marlboro

update

Change 74133 on 2003/01/02 by gregs@gregs_r400_win_marlboro

update

Change 74124 on 2003/01/02 by gregs@gregs_r400_win_marlboro

< ROM straps re-arranged >

Change 74109 on 2003/01/02 by csampayo@fl_csampayo_lt_r400

Some housekeeping

Change 74083 on 2003/01/02 by sbagshaw@sbagshaw

Added description of Scaler double buffered registers in section 9.5, section 7 started with description of functionality and controls of DAC output interface

Change 74027 on 2003/01/01 by gregs@laptop1

update

Change 73990 on 2003/01/01 by gregs@gregs_r400_win_cc

xyz

Change 73877 on 2002/12/31 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 73655 on 2002/12/29 by nbarbier@nbarbier_r400_win_tor

Added Stress test for dispout.

Change 73552 on 2002/12/27 by lchen@lchen_crayola0

fix a typo

Change 73548 on 2002/12/27 by lchen@lchen_crayola0

update the r400 MEM IO spec based on latest info

Change 73425 on 2002/12/26 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 73424 on 2002/12/26 by mkelly@fl_mkelly_r400_win_laptop

Update tracker...

Change 73328 on 2002/12/24 by gregs@gregs_r400_win_cc

added iodft modules configuration in column AC + modifications in VBA write pad_data routine.

Change 73234 on 2002/12/23 by enewman@enewman_r400_linux_marlboro

updated for NL 3.0

Change 72971 on 2002/12/20 by jhoule@MA_JHOULE

Minor corrections and additions to the LOD calculation

Change 72964 on 2002/12/20 by jhoule@MA_JHOULE

v0.9.16
Better description of LOD computation.
Updated pseudo-code.
Update TP_TC interface (pitches).

Change 72948 on 2002/12/20 by nbarbier@nbarbier_r400_win_tor

Added Genlocking test to dispout stress test section.

Change 72807 on 2002/12/20 by gabarca@gabarca_crayola_win_cvd

fixed viewport start spec

Change 72717 on 2002/12/20 by gregs@gregs_r400_win_marlboro

rev 3 netlist

Change 72710 on 2002/12/20 by smoss@smoss_crayola_win

SU tests

Change 72657 on 2002/12/19 by csampayo@fl_csampayo_r400

Updated ststus for tests:
r400vgt_real_time_events_04
r400vgt_real_time_events_05

Change 72591 on 2002/12/19 by beiwang@bei_depot

Copied over white box testing item from testenv/verification/MC MH Test Plan.doc

Added in color preliminary thoughts of how to implement these tests/checks/monitors.

Change 72541 on 2002/12/19 by ashishs@fl_ashishs_r400_win

update

Change 72527 on 2002/12/19 by smoss@smoss_crayola_win

SU tests

Change 72526 on 2002/12/19 by jasif@jasif_r400_win_tor

Added mono_colour_reg_test, misc_reg_test, ovsc_col_sel1, ovsc_col_sel2, and ovsc_col_sel3.

Change 72483 on 2002/12/19 by jiezhou@cn_jiezhou

small update

Change 72481 on 2002/12/19 by mdoggett@mdoggett_r400_win_platypus

Changed formats 50, 51, 52, 53 in format conversion table. Added format 51 to source address table.
Added 3D linear L1 Tag, removed 3D noise L1 Tag.

Change 72477 on 2002/12/19 by jhoule@MA_JHOULE

Renamed lod to lod_comp in the pseudocode

Change 72429 on 2002/12/19 by csampayo@fl_csampayo_lt_r400

Closed bug# 70

Change 72392 on 2002/12/19 by kcorrell@kcorrell_r400_docs_marlboro_nb

updated implementation description

Change 72323 on 2002/12/18 by gregs@laptop1

added speed sensor register

Change 72297 on 2002/12/18 by gregs@laptop1

re-arranged analog and other display pads

Change 72284 on 2002/12/18 by jacarey@fl_jcarey_desktop

AlphaBlend PM4 Packet Update

1. Microcode Updates
2. Documentation Updates
3. Unit Test

Change 72262 on 2002/12/18 by jhoule@MA_JHOULE

LOD computation update, with more complete anisotropy.
Explanation of clamping, which must be done after LOD biases are applied, itself done after all of the specified pseudo-code.

Also updated some Visio links, but some are screwed up =(

Change 72248 on 2002/12/18 by jacarey@fl_jcarey_desktop

Add note for holding "event triggered" on the visio diagram.

Change 72164 on 2002/12/18 by jacarey@fl_jcarey_desktop

Add register to arm signal to mask false falling/rising edge triggering.

Change 72149 on 2002/12/18 by smoss@smoss_crayola_win

SU tests

Change 72082 on 2002/12/18 by csampayo@fl_csampayo_r400

Updated test status and test_list for the following tests:
r400cl_clip_edgeflags_frustum_corners_01
r400cl_clip_edgeflags_frustum_corners_02

Change 72057 on 2002/12/18 by jasif@jasif_r400_win_tor

Updated.

Change 71957 on 2002/12/17 by gregs@gregs_r400_win_marlboro

new order of Toronto pads

Change 71955 on 2002/12/17 by gregs@gregs_r400_win_marlboro

added DEBUG_legacy_test_en signal

Change 71837 on 2002/12/17 by gregs@gregs_r400_win_marlboro

update

Change 71819 on 2002/12/17 by smoss@smoss_crayola_win

SU tests

Change 71728 on 2002/12/17 by lseiler@lseiler_r400_win_marlboro

Fixed a bug in address.c for computing 3D tiled addresses (the spec had it right) and in finding the 3d Y address for a device address (both address.c and the spec were wrong)

Change 71718 on 2002/12/17 by csampayo@fl_csampayo_r400

Some housekeeping

Change 71667 on 2002/12/17 by jacarey@fl_jcarey_desktop

Add AA_Font Micrococode
Add associated unit-level test
Update PM4 Spec for AA_Font and AlphaBlend PM4 Packets

Change 71577 on 2002/12/16 by vgoel@fl_vgoel2

added bug 957

Change 71575 on 2002/12/16 by scroce@scroce_r400_win_marlboro

Updated for new tests written

Change 71562 on 2002/12/16 by gregs@gregs_r400_win_marlboro

update

Change 71536 on 2002/12/16 by csampayo@fl_csampayo_r400

Added Bug# 101 and 102

Change 71459 on 2002/12/16 by jiezhou@cn_jiezhou

make the viewport size smaller

Change 71390 on 2002/12/16 by rbell@rbell_crayola_win_cvd

More updates for full chip sims

Change 71336 on 2002/12/15 by scroce@scroce_r400_home

Updated tests that were recently written

Change 71334 on 2002/12/15 by gregs@laptop1

fixed DEF file generation

Change 71291 on 2002/12/15 by gregs@laptop1

added fake connections to memory pads

Change 71194 on 2002/12/13 by ashishs@fl_ashishs_r400_win

updated for change # 71014 by mmang

Change 71184 on 2002/12/13 by csampayo@fl_csampayo_lt_r400

Updated status for the following tests:
r400su_polymode_culling_face_01
r400su_polymode_culling_face_02
r400su_polymode_lines_degen_triangle_03

Change 71169 on 2002/12/13 by rfisette@rfisette

Updated TST block spec to reflect ports in the IODFT block

Change 71138 on 2002/12/13 by gregs@gregs_r400_win_marlboro

IO<->MC interface cleaned (the interface is not functional).

Change 71137 on 2002/12/13 by gregs@gregs_r400_win_marlboro

< ROM_ON_VIP straps changed >

Change 70998 on 2002/12/13 by rherrick@ma_rherrick_crayola

New deposit... Turning over to Steve for an edit...

Change 70939 on 2002/12/13 by jiezhou@cn_jiezhou

Add in SCL stress tests

Change 70929 on 2002/12/13 by jacarey@fl_jcarey_desktop

Clarify that "dummy" dwords in the ib_prefetch* packets are set to 0xdeadbeef by the Pre-Fetch Parser.

The emulator is being updated to match the RTL.

Change 70842 on 2002/12/12 by csampayo@fl_csampayo3_r400

Some housekeeping

Change 70691 on 2002/12/12 by smoss@smoss_crayola_win

update for new tests

Change 70670 on 2002/12/12 by jiezhou@cn_jiezhou

Add detailed discription for Frame rate conversion test "SCL_VIDCAP_frame_rate_conv"

Change 70564 on 2002/12/11 by gregs@laptop1

ROM_ON_VIP straps changed to 0011

Change 70479 on 2002/12/11 by mzhu@mzhu_crayola_win_tor

Add description for CRTC_TRIG_OCCURRED and CRTC_TRIG_INTERRUPT

Change 70453 on 2002/12/11 by vgoel@fl_vgoel2

added bug 928 and closed bug 898

Change 70436 on 2002/12/11 by jhoule@MA_JHOULE

Described serialize heuristic for both ALU and TEX instructions.

Change 70411 on 2002/12/11 by rbell@rbell_crayola_win_cvd

Added EXCLUDE_PRIMLIB for primlib tests

Change 70337 on 2002/12/11 by rbell@rbell_crayola_win_cvd

Removed HDCP env var settings

Change 70324 on 2002/12/11 by mdoggett@mdoggett_r400_win_platypus

Updated format 29 in L2 cacheline format conversion table required adding new SM10 and SM11.

Change 70285 on 2002/12/11 by ashishs@fl_ashishs_r400_win

updated count for frustum clip block. CHECKPOINT for test tracker for CL/VTE

Change 70281 on 2002/12/11 by ashishs@fl_ashishs_r400_win

updated the test tracker. Increased the count for blocks where new tests were added. As of this checking the tests required match up the tests required in the approach plan.

Change 70269 on 2002/12/11 by rherrick@ma_rherrick_crayola

Updated Register Write section...

Change 70230 on 2002/12/10 by gregs@gregs_r400_win_marlboro

update

Change 70229 on 2002/12/10 by gregs@gregs_r400_win_marlboro

<ROM_ON_VIP strap is 0101 >

Change 70221 on 2002/12/10 by csampayo@fl_csampayo_r400

Updated status of VGT tests r400vgt_real_time_events_02, _03
Some housekeeping

Change 70128 on 2002/12/10 by gregs@gregs_r400_win_marlboro

update

Change 69936 on 2002/12/10 by jowang@jowang_R400_win

1 test in manual ratio / accum init
2 tests in CRC generation
1 test in mode change
1 test in black pixel / line generation

Change 69929 on 2002/12/10 by scroce@scroce_r400_win_marlboro

Added test lists

Change 69887 on 2002/12/10 by rherrick@ma_rherrick_crayola

Turning it over to Steve for the next update...

Change 69867 on 2002/12/10 by moev@P4CLIENT=moev_r400_sun_marlboro

Spread sheet describing port interfaces between system blocks,
TST block and IO.

Change 69865 on 2002/12/10 by rherrick@ma_rherrick_crayola

New update including Multiple Client random test coverage

Change 69847 on 2002/12/10 by scroce@scroce_r400_home

Added some test names

Change 69780 on 2002/12/09 by rherrick@ma_rherrick_crayola

More details included... Still missing multiple client random section and a review of the DRAM/RBBM features...

Change 69736 on 2002/12/09 by scroce@scroce_r400_win_marlboro

Added test tallies

Change 69684 on 2002/12/09 by rthambim@rthambim_r400_win_tor

Initial revision.

Change 69683 on 2002/12/09 by rthambim@rthambim_r400_win_tor

Updated the strap table.

Change 69583 on 2002/12/09 by rherrick@ma_rherrick_crayola

Checkpoint on the MC testplan for Steve Croce to use...

Change 69544 on 2002/12/08 by gregs@laptop1 removed MODE_DDC2CLK

Change 69460 on 2002/12/07 by gregs@laptop1

added comments on ROM_PORT usage.

Change 69270 on 2002/12/06 by mdoggett@mdoggett_r400_win_platypus

Table of contents, figures and tables updated.

Change 69217 on 2002/12/06 by gregs@laptop1

added delay chains

Change 69208 on 2002/12/06 by mdoggett@mdoggett_r400_win_platypus

Version 0.36.
Corrected labeling of SA10 to SA9.
Corrected positions of sectors for SM7 and SM8.
Updated TCA, TCB and TCO block diagrams and descriptions to match current hardware.

Change 69151 on 2002/12/06 by rherrick@ma_rherrick_crayola

Update environment spec to be closer to reality... Included randomization controls section...

Change 69066 on 2002/12/06 by jacarey@fl_jcarey2

Clarify Ring Buffer Size in DWORDs.

Change 68997 on 2002/12/05 by gregs@gregs_r400_win_marlboro

update

Change 68985 on 2002/12/05 by vgoel@fl_vgoel2

added bug 898

Change 68982 on 2002/12/05 by rthambim@rthambim_r400_win_tor

Initial revision.

Change 68965 on 2002/12/05 by lchen@lchen_crayola0

fix the DIFFSTR name

Change 68962 on 2002/12/05 by lchen@lchen_crayola0

update the spec based on discussions with BOB

Change 68961 on 2002/12/05 by jasif@jasif_r400_win_tor

Updated register settings of vcountBy2 and seqpclkby2 testcases.

Change 68928 on 2002/12/05 by ashishs@fl_ashishs_r400_win

added bug for r400cl_edgeflags_05/06/07

Change 68917 on 2002/12/05 by vgoel@fl_vgoel2

updated to added bug 897

Change 68908 on 2002/12/05 by ashishs@fl_ashishs_r400_win

updated

Change 68825 on 2002/12/05 by dglen@dglen_r400

Added 30 bpp option to progressive YPbPr timings

Change 68775 on 2002/12/05 by ashishs@fl_ashishs_r400_win

cancelled 2 tests from the Approach plan and updated tracker since were redundant tests :
1. r400cl_frustum_simple_01(tcl_clip_frustum_simple_01)
2. r400cl_frustum_simple_02(tcl_clip_frustum_simple_02)

Change 68762 on 2002/12/05 by scamlin@scamlin_crayola_win

12/4/2002 area update for orlando blocks

Change 68506 on 2002/12/04 by rherrick@ma_rherrick_crayola

I hope I waited long enough for the file to write out befire submitting... (I didn't last time)...

Change 68503 on 2002/12/04 by rherrick@ma_rherrick_crayola

Updated Table of Contents...

Change 68499 on 2002/12/04 by rherrick@ma_rherrick_crayola

First pass at permutations of surface accesses...

Change 68439 on 2002/12/04 by rherrick@ma_rherrick_crayola

Beginning of MH Testplan... NOT READY FOR CONSUMPTION!!

Change 68428 on 2002/12/04 by jacarey@fl_jcarey_desktop

Fix Typo in Cond_Write Packet in PM4 Spec.

Change 68327 on 2002/12/04 by jacarey@fl_jcarey_desktop

Update Alignment of the Ring, Indirects, and Real-Time Bases

Change 68323 on 2002/12/04 by rherrick@ma_rherrick_crayola

Further described the DC client characterization with the MH queues and RB queues...

Change 68322 on 2002/12/04 by rherrick@ma_rherrick_crayola

Feedback from Testplan review (December 3, 2002) incorporated into document... Sections still needing detail include the DRAM Tuning register verification support, power management verification support, and IKOS parameter verification support..

Change 68167 on 2002/12/03 by frising@ma_frising

v.0.99d

-Update FLOOR opcodes to match r400 shaders.doc v.1.91.
-Updated LIT macro

Change 68164 on 2002/12/03 by marklee@marklee_crayola

2nd attempt at changing permissions

Change 68162 on 2002/12/03 by marklee@marklee_crayola

delete

Change 68160 on 2002/12/03 by marklee@marklee_crayola

attempt to change permissions

Change 68148 on 2002/12/03 by gregs@gregs_r400_win_marlboro

update

Change 68144 on 2002/12/03 by frising@ma_frising

v.1.91

-FLOOR opcode was not producing correct results for negative 'integer' inputs.

Changed from:

```
FLOOR:

If (SrcA < 0.0f)
  Result = TRUNC(SrcA) + -1.0f;
Else
  Result = TRUNC(SrcA)

To:
FLOOR:

Result = TRUNC(SrcA)
If ( (SrcA < 0.0f) && (SrcA != Result) )
  Result += -1.0f;
```

-Note that emulator was calling math library floor function so this should not be an emulation issue.

Change 68140 on 2002/12/03 by marklee@marklee_crayola

check in global_clocks diagram for Eric N.

Change 68139 on 2002/12/03 by marklee@marklee_crayola

check in this file for Eric N.

Change 68133 on 2002/12/03 by gregs@gregs_r400_win_marlboro

update

Change 68062 on 2002/12/03 by jiezhou@cn_jiezhou

change DMIF stress tests active size from 2560x64 to 2560x16

Change 68048 on 2002/12/03 by scroce@scroce_r400_win_marlboro

Added a line

Change 68034 on 2002/12/03 by rherrick@ma_rherrick_crayola

Initial Deposit... Describes Black Box MC Verification requirements...

Change 67940 on 2002/12/02 by mearl@mearl_r400_win

fixed the offset problem

Change 67933 on 2002/12/02 by frising@ma_frising

v.1.90

-While we don't allow CLI relative addressing into the export file on r400, future chips based on r400 may. This check-in moves the export masking behavior bit (bit[6] of vector destination pointer) to bit[14] (bit[6] of scalar destination pointer). Bit[6] of vector destination pointer now controls logical vs. CLI relative addressing into the register or export file. When exporting, this is a Must Be Zero (logical) field for for software on r400. This will provide binary compatibility.

-Software will need to coordinate this change with emulator release.

Change 67917 on 2002/12/02 by ctaylor@fl_ctaylor_r400_win_marlboro

Add Multipass Pixel Shader Description.

Change 67873 on 2002/12/02 by kcorrell@kcorrell_r400_docs_marlboro_nb

update, especially in the implementation section

Change 67866 on 2002/12/02 by jiezhou@cn_jiezhou

initial release

Change 67842 on 2002/12/02 by mkelly@fl_mkelly_r400_win_laptop

Delete test requirement for rect_v0-v3, covered in CP legacy tests

Change 67834 on 2002/12/02 by jacarey@fl_jcarey_desktop

Remove ME_HALT and ALU_COUT32 Booleans

Change 67779 on 2002/12/02 by ashishs@fl_ashishs_r400_win

updated

Change 67702 on 2002/11/30 by gregs@gregs_r400_win_cc

W45B512/012, SST45LF010, and ROM on VIP straps changed.

Change 67647 on 2002/11/30 by gregs@gregs_r400_win_cc

updated memory clock domain - work in progess.

Change 67590 on 2002/11/29 by sbagshaw@sbagshaw

added new document sections and reorganized existing document to new section headings

Change 67572 on 2002/11/29 by dglen@dglen_r400

Added some IBM Bertha panel cases

Change 67519 on 2002/11/28 by dglen@dglen_r400

Added lots of detail for all TV timings, all panel and projector timings except IBM T221.

Change 67376 on 2002/11/27 by bbloemer@ma_bbloemer

Update.

Change 67343 on 2002/11/27 by tshah@fl_tshah

typo fix in EXTERN_TRIG_CNTL register

Change 67276 on 2002/11/27 by jhoule@MA_JHOULE

Updated with more optimal values (easier precision to reach in hardware).

Change 67275 on 2002/11/27 by jhoule@MA_JHOULE

Updated packing scheme.

Change 67273 on 2002/11/27 by mkelly@fl_mkelly_r400_win_laptop

Verify SU_SC_MODE persp corr disable

Change 67258 on 2002/11/27 by bbloemer@ma_bbloemer

First draft of IO pad functional spec.

Change 67215 on 2002/11/27 by jowang@jowang_R400_win

updated start phase

Change 67172 on 2002/11/27 by smoss@smoss_crayola_win

update

Change 67066 on 2002/11/26 by gregs@gregs_r400_win_marlboro

update

Change 67058 on 2002/11/26 by frising@ma_frising

v.1.60
-add MAG_ANISO_WALK and MIN_ANISO_WALK fields to texture constant to conform to D3D.
Here's how it works:

MAG_ANISO_WALK controls if aniso walk is done when anisotropy filter is enabled and magnifying.
MIN_ANISO_WALK controls if aniso walk is done when anisotropy filter is enabled and minifying.

ANISO_FILTER still controls if anisotropy is enabled. When enabled the minor axis controls the LOD calculation so having MAG_ANISO_WALK and MIN_ANISO_WALK disabled while having ANISO_FILTER enabled is allowed. When ANISO_FILTER is disabled, MAG_ANISO_WALK and MIN_ANISO_WALK are ignored.

This allows the anisotropy walk to be completely orthogonal to mag and mig filters meaning we support anisotropy walking with point, linear and arbitrary filters min/mag filters.

This is actually more powerful than the D3D API. D3D drivers should program the HW as follows to be compliant with the refrast:

```
if ( (mag==ansio) || (min==aniso) ) {
  ANSIO_FILTER = enabled, max set as specified.

  if (mag==aniso) {
    MAG_ANISO_WALK = enabled;
    MAG_FILTER = linear;
  } else {
    MAG_ANISO_WALK = disabled;
    // program MAG_FILTER as usual
  }

  if (min==aniso) {
    MIN_ANISO_WALK = enabled;
    MIN_FILTER = linear;
  } else {
    MIN_ANISO_WALK = disabled;
    // program MIN_FILTER as usual
  }

} else {
  ANISO_FILTER = disabled;
  // program MAG_FILTER and MIN_FILTER as usual
}
```

--
Other notes:
-MAG_ANISO_WALK and MIN_ANISO_WALK are not available in instruction word and must be programmed through constant.
-MAG_ANISO_WALK and MIN_ANISO_WALK are forced to 1 (enabled) when anisotropy is enabled for FetchMultiSample.

Change 66971 on 2002/11/26 by gabarca@gabarca_crayola_win_cvd

The prevention of negatifve oversan is only done if one or both displays are in VGA timing

Change 66688 on 2002/11/25 by mkelly@fl_mkelly_r400_win_laptop

Update SC HW coords tests...

Change 66670 on 2002/11/25 by gabarca@gabarca_crayola_win_cvd

VGA DISP interface signals change to avoid negative overscan

Change 66640 on 2002/11/25 by jasif@jasif_r400_win_tor

Updated.

Change 66603 on 2002/11/25 by ashishs@fl_ashishs_r400_win

updated

Change 66550 on 2002/11/24 by gregs@laptop1

update

Change 66367 on 2002/11/22 by csampayo@fl_csampayo_r400

Updated test list and test tracker for the following tests:
r400vgt_real_time_events_01
r400vgt_real_time_events_02

Change 66356 on 2002/11/22 by gregs@gregs_r400_win_marlboro

Friday update

Change 66347 on 2002/11/22 by sbagshaw@sbagshaw

Revised 0.5 DC System testplan based on feedback from meetings and
wrote detailed test descriptions and procedures for many tests.

Change 66283 on 2002/11/22 by jiezhou@cn_jiezhou

Add two more tests and more parameter settings.

Change 66270 on 2002/11/22 by rfisette@rfisette_r400_sun_marlboro

Design specification for the Test Controller (TST) block.

Change 66256 on 2002/11/22 by gregs@gregs_r400_win_marlboro

<fixed R300 number of instances value >

Change 66176 on 2002/11/22 by jacarey@fl_jcarey_desktop

Update Address Mask for Reg_RMW PM4 Packet.
Associated Documentation Updated

Change 66168 on 2002/11/22 by mkelly@fl_mkelly_r400_win_laptop

Close a bug...

Change 66127 on 2002/11/21 by gabarca@gabarca_crayola_win_cvd

de skew should only affect h disp start

Change 66124 on 2002/11/21 by gregs@gregs_r400_win_marlboro

update

Change 66118 on 2002/11/21 by dglen@dglen_r400

Updated with some new modes for digital TV.
Added V start phase for progressive.

Change 66089 on 2002/11/21 by csampayo@fl_csampayo_r400

Added bug# 97

Change 66088 on 2002/11/21 by jowang@jowang_R400_win

updated with a column for required # of taps

Change 66034 on 2002/11/21 by mzhu@mzhu_crayola_win_tor

Add LB_DATA_GAP_BETWEEN_CHUNKS

Change 66006 on 2002/11/21 by jacarey@fl_jcarey_desktop

Update to Reg_RMW Packet...Streamlined Thanks to Harry...

Change 65997 on 2002/11/21 by gregs@gregs_r400_win_marlboro

update

Change 65972 on 2002/11/21 by jiezhou@cn_jiezhou

minor text updated

Change 65964 on 2002/11/21 by mkelly@fl_mkelly_r400_win_laptop

Refresh for tommorow's status meeting...

Change 65958 on 2002/11/21 by frising@ma_frising

v.1.59
-A marco that computed texture constant offsets somehow got corrupted with the 1.58 check-in causing offsets to be off by one. Folks should move to this version immediately. Sorry. Thanks to John Carey for catching this.

Change 65950 on 2002/11/21 by jacarey@fl_jcarey_desktop

Add Reg_RMW (Read/Modify/Write) to the PM4 Packets
Documentation Update
Associated Unit-Level Test

Change 65863 on 2002/11/21 by jacarey@fl_jcarey_desktop

Revert B7 2D Boolean Setting.
It is only set based on the src_type==5

1. Emulator Update to CP
2. Verilog Update to Micro Engine
3. Remove Unit Test
4. Update CP Documentation

Change 65731 on 2002/11/20 by dglen@dglen_r400

Minor clean up

Change 65722 on 2002/11/20 by dglen@dglen_r400_dell

Major update of formulas.
Corrected many TV timings.

Change 65679 on 2002/11/20 by sbagshaw@sbagshaw

added block select addresses for BIF debug blocks
added placeholders for lists of BIF debug values on debug bus

Change 65637 on 2002/11/20 by jacarey@fl_jcarey_desktop

Rename two fields in the RBBM_Status Register.

```
        Block Name  Machine   #Tests  #ExpectedRuns   #Runs   #Fail   #Pass  #NotRun
#NoGold   #NoRef   #Error   Passing %
        ------------------------------------------------------------------------------------------------------
------
        cl        FL_JOHNC      3        3        3     0     3      0     0       0       0    100.00%
```

```
        cp         FL_JOHNC     12       12       12     0    12      0     0       0       0
100.00%
        perf       FL_JOHNC      3        3        3     0     3      0     0       0       0    100.00%
        quickemu   FL_JOHNC     16       16       16     0    16      0     0       0       0
100.00%
        rb         FL_JOHNC      7        7        7     0     7      0     0       0       0    100.00%
        sc         FL_JOHNC     20       20       20     0    20      0     0       0       0
100.00%
        su         FL_JOHNC      3        3        3     0     3      0     0       0       0    100.00%
        vgt        FL_JOHNC      3        3        3     0     3      0     0       0       0    100.00%
        vte        FL_JOHNC      1        1        1     0     1      0     0       0       0    100.00%
```

Change 65599 on 2002/11/20 by csampayo@fl_csampayo_r400

Some VGT housekeeping

Change 65588 on 2002/11/20 by jacarey@fl_jcarey_desktop

8bpp-to-16bpp TLU Issue for 2D

Change 65570 on 2002/11/20 by rbell@rbell_crayola_win_cvd3

Added CP to the chip_vidfull conf

Change 65561 on 2002/11/20 by rbell@rbell_crayola_win_cvd3

Added a section for Video IP fullchip sims

Change 65539 on 2002/11/20 by rthambim@rthambim_r400_win_tor

Fixed syntax in naming.

Change 65537 on 2002/11/20 by rthambim@rthambim_r400_win_tor

Added 3gio ports.

Change 65500 on 2002/11/20 by jiezhou@cn_jiezhou

initial release

Change 65351 on 2002/11/19 by csampayo@fl_csampayo_r400

Closed bug# 73

Change 65312 on 2002/11/19 by frising@ma_frising

v.0.99c
-change R400_FP_NAN to 0xFFC00000 (from 0x7FC00000) to match intel for indefinite floating point operations.

Change 65310 on 2002/11/19 by vgoel@fl_vgoel2

updated to include bug 788

Change 65307 on 2002/11/19 by vgoel@fl_vgoel2

updated for number of HOS tests written

Change 65303 on 2002/11/19 by nluu@nluu_r400_doclib_cnnb

- update

Change 65297 on 2002/11/19 by nluu@nluu_r400_doclib_cnnb

- update

Change 65274 on 2002/11/19 by mkelly@fl_mkelly_r400_win_laptop

Update regress_e tests...

Change 65267 on 2002/11/19 by mkelly@fl_mkelly_r400_win_laptop

Update local Orlando bug tracking matrix...

Change 65263 on 2002/11/19 by ashishs@fl_ashishs_r400_win

updated

Change 65254 on 2002/11/19 by vgoel@fl_vgoel2

added second export related bug

Change 65237 on 2002/11/19 by sbagshaw@sbagshaw

Updated DC Debug Bus document to include 3 changes that are necessary for AUTOREG generated block files.
Updated indirect debug register allocations for each major subblock in DC
Updated debug trigger logic descriptions for value and edge pattern triggering functionality

Change 65234 on 2002/11/19 by csampayo@fl_csampayo_r400

Added 3 new VGT tests and updated test_list and test tracker accordingly

Change 65229 on 2002/11/19 by vgoel@fl_vgoel2

added bug 785

Change 65172 on 2002/11/19 by mpersaud@mpersaud_r400_win_tor

Added DCCIF_clk_on description.

Change 65072 on 2002/11/18 by jowang@jowang_R400_win

update with interlaced nomeclature

Change 65066 on 2002/11/18 by gregs@gregs_r400_win_marlboro

<rev 8.0 - dft >

Change 65042 on 2002/11/18 by imuskatb@imuskatb_r400_win_cnimuskatb

updated test doc

Change 64983 on 2002/11/18 by jowang@jowang_R400_win

updated with more M3 tests and testlist

Change 64857 on 2002/11/18 by jacarey@fl_jcarey_desktop

RB_BLENDCONTROL.Color_Dither_Mode is set to DITHER_LUT for GradFill packets
where the DST_TYPE != 32bpp.

Change 64852 on 2002/11/18 by jasif@jasif_r400_win_tor

Updated.

Change 64849 on 2002/11/18 by sbagshaw@sbagshaw

added new column for test type -- chip level or chip and block level
added more test descriptions.

Change 64846 on 2002/11/18 by jasif@jasif_r400_win_tor

Updated.

Change 64833 on 2002/11/18 by jacarey@fl_jcarey_desktop

Add note for ME_RTS generation for read operations initiated by the micro engine.

Change 64829 on 2002/11/18 by ashishs@fl_ashishs_r400_win

added triangles with edgeflags and texture test to the tracker

Change 64701 on 2002/11/16 by gregs@laptop1

update

Change 64590 on 2002/11/15 by donaldl@fl_donaldl_p4

Added sc_packer debug bus (DEBUG_PKR_0).

Change 64571 on 2002/11/15 by gregs@gregs_r400_win_marlboro

update

Change 64546 on 2002/11/15 by sbagshaw@sbagshaw

Debug document modified to include interface to 12 bit debug bus value and enable directly from DC block.

Change 64447 on 2002/11/15 by vgoel@fl_vgoel2

updated for HOS tests written so far

Change 64418 on 2002/11/15 by frising@ma_frising

v.1.89

-show FRACT instructions being implemented as: SRC + -FLOOR(SRC)
-This should now sync the instructions with v.0.99b of numerics doc.

Change 64403 on 2002/11/15 by frising@ma_frising

v.0.99b

-checkpoint
--implement shader pipe instructions is WZYX order.
--fix a couple typos
--start sync to v.1.89 of shader pipe spec.

Change 64389 on 2002/11/15 by jasif@jasif_r400_win_tor

Updated.

Change 64381 on 2002/11/15 by frising@ma_frising

v.1.88

-specify that Result.X of CUBE instruction returns 2.0f * MajorAxis instead of max to avoid confusion. See numerics doc for details of CUBE instruction.

-specify function of all instructions in WZYX order. Nothing changing here; it's just for spec consistency (and happens to reflect the actual order of operations in the shader pipe hardware).

Change 64379 on 2002/11/15 by gabarca@gabarca_crayola_win_cvd

Mode 13, mode 62, mode X have ATTR_PCLKBY2 = 1

Change 64309 on 2002/11/15 by mkelly@fl_mkelly_r400_win_laptop

Log an SC bugzilla on color prob on prim edges...

Change 64144 on 2002/11/14 by jacarey@fl_jcarey_desktop

1. RTL Update to Set 2D Boolean B0 for LUT Color Sources
2. CP Spec Updates for #1
3. CP Spec Update for the CP_DMA_STAT Register

Change 64064 on 2002/11/14 by mkelly@fl_mkelly_r400_win_laptop

Update local, Orlando bug tracking info....

Change 64050 on 2002/11/14 by vgoel@fl_vgoel2

updated

Change 64022 on 2002/11/14 by mkelly@fl_mkelly_r400_win_laptop

Add textured line to regress_e

Change 64000 on 2002/11/14 by gregs@gregs_r400_win_marlboro

< fixed some wire in DVO interface >

Change 63978 on 2002/11/14 by jasif@jasif_r400_win_tor

Updated.

Change 63959 on 2002/11/14 by gabarca@gabarca_crayola_win_cvd

split the DisplayEnable test in three

Change 63920 on 2002/11/14 by grayc@grayc_r400_win

needs to be updated ... will add back in later

Change 63819 on 2002/11/13 by dglen@dglen_r400_dell

Major update to outline. VGA block spec is now best source for details.

Change 63811 on 2002/11/13 by csampayo@fl_csampayo_r400

Added bug# 87

Change 63793 on 2002/11/13 by gregs@gregs_r400_win_marlboro

added CGM clock monitoring pin

Change 63776 on 2002/11/13 by csampayo@fl_csampayo_r400

Closed bug# 86

Change 63751 on 2002/11/13 by jasif@jasif_r400_win_tor

Updated.

Change 63732 on 2002/11/13 by frising@ma_frising

v.1.87

-when using absolute constant addressing all constants in instruction are absolute. This allows compiler to easily perform mov operations on absolute constants.

-Document instruction word in WZYX order (i.e. high bits to low bits). Documentation issue only.

-lots more work cleaning up instruction word documentation (please study).

-It may not have been clear in the past but indexing of exports is not permitted. This update should make that obvious.

-removed following export restriction which no longer applies: '1) When doing a Scalar export of 'pixels' or 'position', only the 'W' component will contain the scalar result. The other 3 components will be expanded to 0.0. When exporting to 'parameters' the scalar result is put into all 4 components.'

Users should just use the scalar and vector destination masks appropriately to achieve whatever result they want.

-when exporting, bit 6 in instruction word now controls masking behavior during parameter exports when both scalar and vector masks for a component are 0. See table 3.2.1.4

Change 63680 on 2002/11/13 by jacarey@fl_jcarey_desktop

Im_Load and Im_Load_Immediate packets write the SQ_PS_PROGRAM register for real-time shader code
updates. This is documented in the PM4 spec now.

Change 63675 on 2002/11/13 by gregs@gregs_r400_win_marlboro

<TEST_MCLK tste bug fixed >

Change 63654 on 2002/11/13 by beiwang@bei_depot

Updated for mc_client_intfc block AGP/Multi-Sample functionality as well as more details for Protocol engine

Change 63653 on 2002/11/13 by mzhu@mzhu_crayola_win_tor

Update the 3rd milestone tests

Change 63604 on 2002/11/13 by donaldl@fl_donaldl_p4

Initial -- Debug bus, bit definitions.

Change 63562 on 2002/11/12 by gregs@laptop1

ccc

Change 63552 on 2002/11/12 by csampayo@fl_csampayo_lt_r400

Updated test_list and test tracker for the following tests:
r400vgt_multi_pass_pix_shader_07
r400vgt_multi_pass_pix_shader_08

Change 63509 on 2002/11/12 by frising@ma_frising

v.1.86

-add note that Constant0 refers to the first constant in the instruction while Constant1 and Constant2 refer to the second and third constants in the instruction respectively.

-added note that the GPR write-back table rules only apply when the scalar and vector destination pointers are the same. This should be obvious, but clarity never hurts.

-when doing an export and both scalar and vector channels are masked a 0.0f is now generated.

-updated Exports Types and Addresses section to match what is in SQ doc.

-tried to clean-up export rules section.

-removed previous vector and previous scalar from source selects in instruction word.

-added clamping code to LOG_CLAMPED instruction.

-fixed a bunch of typos and other misc clean-up including a couple places where I was mixing ABGRs with my WZYXs in the instruction definitions. Tried to be consistent when refering to bits in instruction word.

Change 63500 on 2002/11/12 by csampayo@fl_csampayo_lt_r400

Some housekeeping

Change 63490 on 2002/11/12 by jacarey@fl_jcarey_desktop

Add Write Confirm Signals to CP_STAT Register

Change 63478 on 2002/11/12 by rramsey@RRAMSEY_P4_r400_win

Update quadorder drawing and add it to the sc spec

Change 63454 on 2002/11/12 by ashishs@fl_ashishs_r400_win

update

Change 63447 on 2002/11/12 by mzhu@mzhu_crayola_win_tor

Update 11.16 Cursor_Cntl

Change 63398 on 2002/11/12 by ctaylor@fl_ctaylor_r400_win_marlboro

Adding SC visio and xls files for documentation

Change 63336 on 2002/11/12 by jacarey@fl_jcarey_desktop

Baseline Debug Documents for the CP and RBBM

Change 63285 on 2002/11/11 by peterp@MA_PETE_LT

Added common format to CG, ROM, DBG

Change 63274 on 2002/11/11 by csampayo@fl_csampayo_r400

Closed bug# 83

Change 63262 on 2002/11/11 by gregs@gregs_r400_win_marlboro

backup of drawings

Change 63258 on 2002/11/11 by gregs@gregs_r400_win_marlboro

update

Change 63197 on 2002/11/11 by rramsey@RRAMSEY_P4_r400_win

details intra-tile quad processing order when pa_su_sc_model_cntl.QUAD_ORDER_ENABLE is set

Change 63189 on 2002/11/11 by peterp@MA_PETE_LT

Added common format for MC and MH

Change 63174 on 2002/11/11 by peterp@MA_PETE_LT

Added common format to cp and rbbm

Change 63155 on 2002/11/11 by jacarey@fl_jcarey_desktop

Add CP_RT_STAT register to spec.

Change 63149 on 2002/11/11 by jowang@jowang_R400_win

add full/empty tests

Change 63148 on 2002/11/11 by gabarca@gabarca_crayola_win_cvd

Added VGA_DISP_sync_en

Change 63098 on 2002/11/11 by jacarey@fl_jcarey_desktop

1. Performance Signals for RTEE
2. Split Busy Signal from the CSF into a real-time and non-real-time version

Change 63094 on 2002/11/11 by jacarey@fl_jcarey_desktop

1. Add register to micro engine guts for timing
2. Add FIFOs for Timing in the Synchronization Logic
3. Update CP's Performance Counter Selects in perfcount.doc

Change 63077 on 2002/11/11 by sbagshaw@sbagshaw

Revision 0.35 of R400 DC (Toront) Test Plan for System and Stress tests

Change 63062 on 2002/11/11 by jasif@jasif_r400_win_tor

Updated

Change 63053 on 2002/11/11 by mpersaud@mpersaud_r400_win_tor

Added debug signals and WAIT/FREQUENCY/SIZE values to the dccif interface.

Change 62983 on 2002/11/09 by peterp@MA_PETE_LT

Added common format for Analog and BIF

Change 62925 on 2002/11/08 by jhoule@MA_JHOULE

Latest uRF expand positionings for 16 or 32 bpp post-blender conversion (before fix2float convert, which happens on SP side).

First 2 sheets are current POR.
Others are dropped alternatives.

Change 62910 on 2002/11/08 by csampayo@fl_csampayo_lt_r400

Updated status in test tracker and added to test_list: r400vgt_real_time_events_01

Change 62909 on 2002/11/08 by bbloemer@ma_bbloemer

Added MC MH Test Plan

Change 62868 on 2002/11/08 by csampayo@fl_csampayo_lt_r400

Added bug# 86. Some housekeeping

Change 62857 on 2002/11/08 by mpersaud@mpersaud_r400_win_tor

Added VIP block select and group debug signals.

Change 62813 on 2002/11/08 by jacarey@fl_jcarey_desktop

1. Update CP Spec for Debug Signals
2. Update One of the 2D Unit Tests to Read Back Debug Information

Change 62808 on 2002/11/08 by mzhu@mzhu_crayola_win_tor

Update 3.4.9.4 Color Space Conversion

Change 62771 on 2002/11/08 by gregs@gregs_r400_win_marlboro

update

Change 62759 on 2002/11/08 by mzhu@mzhu_crayola_win_tor

Add DxCRTC_FLOW_CONTROL_POLARITY, DxCRTC_TRIG_INPUT_STATUS and
DxCRTC_TRIG_POLARITY_STATUS register bits

Change 62748 on 2002/11/08 by ashishs@fl_ashishs_r400_win

update

Change 62743 on 2002/11/08 by jacarey@fl_jcarey_desktop

Add RBBM_BIF_int to the RBBM's performance counters.
RTL and Spec Update.

Change 62732 on 2002/11/08 by mpersaud@mpersaud_r400_win_tor

Added state machine diagrams.

Change 62711 on 2002/11/08 by ashishs@fl_ashishs_r400_win

updated for last 3 vte tests viz r400vte_pos_neg_combos_01/02/03

Change 62511 on 2002/11/07 by imuskatb@imuskatb_r400_win_cnimuskatb

updated

Change 62447 on 2002/11/07 by jacarey@fl_jcarey_desktop

Added note to RBBM spec regarding the performance counters.

Change 62444 on 2002/11/07 by jacarey@fl_jcarey_desktop

Updates for the CP_STAT register

Change 62432 on 2002/11/07 by gabarca@gabarca_crayola_win_cvd

added perf count signals

Change 62338 on 2002/11/07 by jasif@jasif_r400_win_tor

Updated

Change 62301 on 2002/11/07 by csampayo@fl_csampayo_r400

Housekeeping update

Change 62266 on 2002/11/07 by jacarey@fl_jcarey_desktop

Updates for Soft Reset to Register Descriptions

Change 62198 on 2002/11/06 by gregs@laptop1

<enter description hupdate
ere>

Change 62131 on 2002/11/06 by gabarca@gabarca_crayola_win_cvd

One test in each set of the VGA to CRTC parameters section should do display capture. Tests in the display section normally do capture

Change 62066 on 2002/11/06 by mpersaud@mpersaud_r400_win_tor

added VIP_DISP_eof_fcp and VIP_DISP_pol_fcp to interface with DISP/CRTC

Change 62062 on 2002/11/06 by peterp@MA_PETE_LT

Added common synthesis report format to DC block. Combined DC and VIP results into the DC results.

Change 62055 on 2002/11/06 by mzhu@mzhu_crayola_win_tor

Update 3.4.9.4 Color Space Conversion

Change 62037 on 2002/11/06 by mzhu@mzhu_crayola_win_tor

Update constant matrix for TVRGB output in 11.18 Matrix Transform and Adjustment.

Change 62033 on 2002/11/06 by lkang@lkang_r400_win_tor

update on SCLK dynamic clocking

Change 61984 on 2002/11/06 by beiwang@bei_depot

Updated the bus interface to MH and RB

Change 61964 on 2002/11/06 by jacarey@fl_jcarey_desktop

Update Soft Reset Description

Change 61936 on 2002/11/06 by gabarca@gabarca_crayola_win_cvd

clarified grph_pack in spreadsheet

Change 61912 on 2002/11/06 by jacarey@fl_jcarey_desktop

Update to GradFill Description in PM4 Spec

Change 61857 on 2002/11/05 by jasif@jasif_r400_win_tor

Change name of overscan test to overscanColourSelect.

Change 61850 on 2002/11/05 by jasif@jasif_r400_win_tor

Updated.

Change 61843 on 2002/11/05 by gregs@gregs_r400_win_marlboro

dft update

Change 61837 on 2002/11/05 by mzhu@mzhu_crayola_win_tor

Update 3.4.9.4 Color Space Conversion

Change 61803 on 2002/11/05 by jacarey@fl_jcarey_desktop

Update format of the Grad_Fill packet per discussion on 11-05-2002.

Change 61761 on 2002/11/05 by dwong@cndwong2

Added details on endian swap and an extra field for indirect buffer swap setting

Change 61728 on 2002/11/05 by gabarca@gabarca_crayola_win_cvd

Added description of VESA mode tests

Change 61698 on 2002/11/05 by imuskatb@imuskatb_r400_win_laptop

updated docs

Change 61664 on 2002/11/05 by gregs@gregs_r400_win_marlboro

adding id module + new straps

Change 61642 on 2002/11/05 by mzhu@mzhu_crayola_win_tor

Add one intermediate result fraction bit for Overlay Matrix Transform in chapter 11.6 and Matrix Transform and Adjustment in chapter 11.18

Change 61620 on 2002/11/05 by mzhu@mzhu_crayola_win_tor

add "Out SRGB->YCbCr" and "Out sRGB->TVRGB" for overlay pixels

Change 61592 on 2002/11/05 by vgoel@fl_vgoel2

updated to close bug 608

Change 61582 on 2002/11/05 by mpersaud@mpersaud_r400_win_tor

Added to source control

Change 61541 on 2002/11/04 by jasif@jasif_r400_win_tor

Updated.

Change 61526 on 2002/11/04 by rfevreau@rfevreau_r400_win

X2 Mag tests

Change 61518 on 2002/11/04 by jowang@jowang_R400_win

added back-to-back writes to HOST

Change 61484 on 2002/11/04 by dwong@cndwong2

added in performance counter

Change 61480 on 2002/11/04 by jacarey@fl_jcarey_desktop

Fix Typo in MPEG_INDEX packet documentation.

Change 61470 on 2002/11/04 by peterp@MA_PETE_LT

VGT, PA and SC updated with common format for area and synthesis date

Change 61467 on 2002/11/04 by grayc@grayc_r400_win

better debug statements

Change 61453 on 2002/11/04 by jacarey@fl_jcarey_desktop

Update MPEG Index Packet
1. Remove "Dummy" and "Mask" DWORDs
2. RectList primtype is assumed, so CP only outputs 3 indices (Original +2)

Change 61449 on 2002/11/04 by jasif@jasif_r400_win_tor

Updated

Change 61444 on 2002/11/04 by peterp@MA_PETE_LT

Updated SQ with common format of reporting area and synthesis date

Change 61429 on 2002/11/04 by bbloemer@ma_bbloemer

Improved figure insertion, I hope.

Change 61416 on 2002/11/04 by jacarey@fl_jcarey_desktop

Update the Microcode RAM Size(s)

Change 61403 on 2002/11/04 by ashishs@fl_ashishs_r400_win

updated

Change 61360 on 2002/11/04 by imuskatb@imuskatb_r400_win_cnimuskatb

updated

Change 61342 on 2002/11/04 by frising@ma_frising

v.1.58
-At request of SW merged TYPE/STATE fields in constants. These are SW only bits used by PM4 capture utilities.
-Fixed 3D size typo, had width, height, height instead of width, height, depth
-Make addressing mode names consistent with SP; now have logical and current loop index relative addressing
-Mark R400_DATA_FORMAT 63 as reserved
-Clean up and clarify R400_DATA_FORMAT Notes sections based on feedback

Change 61310 on 2002/11/04 by jacarey@fl_jcarey_desktop

Update per Lili Sinclair's E-mail.

Change 61187 on 2002/11/01 by mdoggett@mdoggett_r400_win_platypus

Added degamma dxt. Updated format table with new _as_16_16_16_16 formats. Updated L2 cacheline format conversion table. Added degamma dxt section with equations for calculating different dxt interpolations.

Change 61163 on 2002/11/01 by ashishs@fl_ashishs_r400_win

updated

Change 61151 on 2002/11/01 by frising@ma_frising

v.1.85
-clarifed how constant and register addressing works. For nomenclature I settled on absolute, logical and relative addressing. New tables added.
-documented how constant2 works.
-Wrote a blurb and added a table on GPR write-back precedence.
-Cleaned up ALU instruction format table

Change 61074 on 2002/11/01 by vgoel@fl_vgoel2

updated tracker for closed bugs

Change 61068 on 2002/11/01 by gregs@gregs_r400_win_marlboro

update

Change 61055 on 2002/11/01 by gabarca@gabarca_crayola_win_cvd

Added viewport X and Y start

Change 61036 on 2002/11/01 by jacarey@fl_jcarey2

Update RBBM's Performance Counter Signal List

Change 60996 on 2002/11/01 by nbarbier@nbarbier_r400_win_tor

Updated GENERICA & GENERICB muxes.
Added GENERICC pad to interface.

Change 60992 on 2002/11/01 by peterp@MA_PETE_LT

Synthesis results from SP/SX

Change 60988 on 2002/11/01 by rthambim@rthambim_r400_win_tor

Updated the strap table.

Change 60985 on 2002/11/01 by dwong@cndwong2

Added in xDCT performance counter details

Change 60981 on 2002/11/01 by rthambim@rthambim_r400_win_tor

Added new clock enable signal - BIF_VGA_busy.

Change 60937 on 2002/11/01 by mkelly@fl_mkelly_r400_win_laptop

Broken out line list cases from parameterized test
Update tracker

Change 60934 on 2002/11/01 by mpersaud@mpersaud_r400_win_tor

Rev 0.7 - Update port names to CP

Change 60913 on 2002/11/01 by peterp@MA_PETE_LT

Started entry of date field for synthesis results - corrected RB/RC area

Change 60910 on 2002/11/01 by jacarey@fl_jcarey2

Update CP's Performance Monitoring Signals.

Change 60859 on 2002/10/31 by beiwang@bei_depot

Updated Protocol Engine Drawing

Change 60850 on 2002/10/31 by vgoel@fl_vgoel2

updated with new bugs and closed dates for some bugs

Change 60825 on 2002/10/31 by gregs@gregs_r400_win_marlboro

update

Change 60810 on 2002/10/31 by csampayo@fl_csampayo_r400

Added new SU test, updated test_list and test tracker accordingly

Change 60808 on 2002/10/31 by bbloemer@ma_bbloemer

Updates for the design review.

Change 60786 on 2002/10/31 by mzhu@mzhu_crayola_win_tor

Update for re-organizing the MH data in read_buff module.
Add MH-DC interface signal list, MH return data format and DMIF output data format in Appendix

Change 60768 on 2002/10/31 by jacarey@fl_jcarey_desktop

Fix Name of Signal From the RBBM.

Change 60738 on 2002/10/31 by hartogs@fl_hartogs

Added section for determining the required input data size based on the draw initiator and the grouper programming registers.
Added section on general sanity checks for grouper programming in major mode 1.
Updated event enumeration to match block file (added several events).
Added DI_PT_2D_TRI_STRIP to the prim type enumeration in the VGT_DRAW_INITIATOR register.
Added VGT_GRP_2D_TRI enumeration to prim type field of VGT_GROUP_PRIM_TYPE register (and deleted some unused prim type enumerations).
Updated Major Mode 0 settings table for 2D compond index changes. Added 2D_TRI_STRIP entry to this table.
Changed address of many of the GFX registers registers to match addresses in the block file.
Added VGT_MULTI_PRIM_IB_RESET_INDX register with description.
Changed VGT_VTX_TIMEOUT_REG to VGT_ VGT_VTX_VECT_EJECT_REG register
Added VGT_DMA_DATA_FIFO_DEPTH register· Added VGT_DMA_REQ_FIFO_DEPTH register
Added VGT_DRAW_INIT_FIFO_DEPTH register
Added VGT_LAST_COPY_STATE register.
Added section called "Draw Initiator Programming" containing two subsections: "Number of Indices" and "Using the Multi-prim Index Buffer Reset Functionality"

Change 60701 on 2002/10/31 by georgev@ma_georgev

Put in documentation for floating point numbers.

Change 60669 on 2002/10/31 by gabarca@gabarca_crayola_win_cvd

ODD_EVEN_MD_PGSEL is 1 in all modes

Change 60634 on 2002/10/31 by mpersaud@mpersaud_r400_win_tor

Rev 1.1 Mahendra PersaudDate: Oct 31, 2002
Added VIP signals to CP
Moved and renamed document to ...\doc_lib\design\blocks\chip\R400 - DC_CP Interface.doc

Change 60628 on 2002/10/31 by smoss@smoss_crayola_win

su tests

Change 60627 on 2002/10/31 by mzhu@mzhu_crayola_win_tor

Add dual display tests in 3.4.9.12

Change 60624 on 2002/10/31 by khabbari@khabbari_r400_win

disp signals added to disp/cp inteface doc

Change 60620 on 2002/10/31 by mkelly@fl_mkelly_r400_win_laptop

Update screen scissor and window scissor tests for tracking...

Change 60565 on 2002/10/30 by gregs@laptop1

update

Change 60393 on 2002/10/30 by jacarey@fl_jcarey2

Fix Typo in CP_PERFMON_CNTL register description

Change 60376 on 2002/10/30 by mpersaud@mpersaud_r400_win_tor

Updated with Appendix A - MH_DCC Interface

Change 60360 on 2002/10/30 by gregs@laptop1

added IO_RBBM_genericc_y signal

Change 60295 on 2002/10/30 by mzhu@mzhu_crayola_win_tor

Update 3.4.9.4 Color Space Conversion
Add 3.4.9.10 double buffer registers
Add 3.4.9.11 DCP CRC

Change 60294 on 2002/10/30 by rbell@rbell_crayola_win_cvd

updated

Change 60264 on 2002/10/30 by frising@ma_frising

v.1.8
-bring up to date with v.0.99a of R400numerics.doc. Includes adding new instructions, updating existing instructions, and moving to counter based predicate scheme.

Change 60178 on 2002/10/29 by frising@ma_frising

v.0.99a

-Sync up with v.1.8 of shaders.doc that I've been editing and will check-in soon.

Change 60167 on 2002/10/29 by gregs@gregs_r400_win_marlboro

update

Change 60126 on 2002/10/29 by rbell@rbell_crayola_win_cvd

Finished MS3 D2 tests

Change 60112 on 2002/10/29 by vgoel@fl_vgoel2

added rpatch bug for tessellation 11.5

Change 60078 on 2002/10/29 by jowang@jowang_R400_win

added CRC generation

Change 59994 on 2002/10/29 by jacarey@fl_jcarey2

Add registers for the CP Performance Counter

Change 59989 on 2002/10/29 by jacarey@fl_jcarey2

RBBM Performance Counters

Change 59944 on 2002/10/29 by csampayo@fl_csampayo_r400

Added bug# 83

Change 59941 on 2002/10/29 by llefebvr@llefebvre_laptop_r400

backup and SQ->SP interface change.

Change 59932 on 2002/10/29 by nluu@nluu_r400_doclib_cnnb

- update interface to add start_of_cycle and op code signals

- cycle is 1 clock shorter, strobe is asserted 1 clock after data is driven instead of 2 clocks
- decodes are don't cares when not in active cycle
- ready should be high by default

Change 59930 on 2002/10/29 by gregs@gregs_r400_win_marlboro

added Test Controller connections to IO DFT cells.

Change 59904 on 2002/10/29 by jacarey@fl_jcarey_desktop

Source clipping is only done by the CP for ROPs that include a source term.

Change 59873 on 2002/10/28 by gregs@gregs_r400_win_marlboro

io dft cells inserted - fixes.

Change 59867 on 2002/10/28 by frising@ma_frising

v.0.99
-updated to counter based predicate instructions.
-lots of work on 32-bit rounding issues. Still more to do especially for 16-bit FP and conversion between formats and things like RF expansions.

Change 59820 on 2002/10/28 by mzhu@mzhu_crayola_win_tor

Add matrix transform constant in 11.18

Change 59816 on 2002/10/28 by rfevreau@rfevreau_r400_win

New goldens

Change 59789 on 2002/10/28 by jacarey@fl_jcarey2

AlphaBlend, AAFONT and Load_Execute marked as "Not Currently Supported"

Change 59706 on 2002/10/28 by csampayo@fl_csampayo_r400

Added bug# 82

Change 59693 on 2002/10/28 by jacarey@fl_jcarey2

Swap bits for oper=gmcdecode flags

Change 59683 on 2002/10/28 by csampayo@fl_csampayo_r400

Try again

Change 59682 on 2002/10/28 by csampayo@fl_csampayo_r400

Added Bugzilla report# to bug# 81

Change 59681 on 2002/10/28 by csampayo@fl_csampayo_r400

Some housekeeping

Change 59677 on 2002/10/28 by jacarey@fl_jcarey2

Add:
1. ROP7:4 != ROP3:0 flag for oper=gmcdecode
2. Src_Clip_Disable Flag for oper=gmcdecode

Change 59592 on 2002/10/27 by csampayo@fl_csampayo_lt_r400

Updated test_list and test tracker status for the following tests:
r400vgt_multi_pass_pix_shader_01
r400vgt_multi_pass_pix_shader_02
r400vgt_multi_pass_pix_shader_03
r400vgt_multi_pass_pix_shader_04
r400vgt_multi_pass_pix_shader_05
r400vgt_multi_pass_pix_shader_06

Change 59437 on 2002/10/25 by ashishs@fl_ashishs_r400_win

update

Change 59405 on 2002/10/25 by lseiler@lseiler_r400_win_marlboro

Added a column for device address

Change 59397 on 2002/10/25 by gregs@gregs_r400_win_marlboro

updated the buscfg strap encodings.

Change 59395 on 2002/10/25 by jacarey@fl_jcarey2

Remove CP_CONTEXT_ID register it will be in the VGT

Change 59371 on 2002/10/25 by jacarey@fl_jcarey2

Update Performance Conections.

Change 59342 on 2002/10/25 by jacarey@fl_jcarey_desktop

Document CP_CONTEXT_ID register.

Change 59317 on 2002/10/25 by jacarey@fl_jcarey2

Update RBBM Debug Register.

Change 59314 on 2002/10/25 by jacarey@fl_jcarey2

Updates to RBBM spec to clarify wait_until, nqwait, and Isync wait conditions.

Change 59255 on 2002/10/24 by grayc@grayc_r400_win

initial release

Change 59211 on 2002/10/24 by ashishs@fl_ashishs_r400_win

update

Change 59151 on 2002/10/24 by frising@ma_frising

v.1.57
- Rework of degamma control to be more in line with hardware schedule :) Perhaps surprisingly, this resulted in a cleaner SW interface and easier HW implementation. This involved:

- Removing of DEGAMMA_CNTL_ALL

- Adding 4 new DATA_FORMATs: FMT_8_8_8_8_AS_16_16_16_16, FMT_DXT1_AS_16_16_16_16, FMT_DXT2_3_AS_16_16_16_16 and FMT_DXT4_5_AS_16_16_16_16.

- Updated DATA_FORMAT table. This is now much cleaner. FOLKS SHOULD STUDY THIS CLOSELY.

- FMT_DXN is no longer a degammable format.

- Update DATA_FORMAT note 7.) to say:
"7.) Channels being degamm'd remain unsigned repeating fraction after degamma. Enabling degamma in any channel does not change the format in which the data is stored in the L2 cache. This enables software to make trade-offs between high quality degamma and performance. Specifically, formats FMT_8_8_8_8, FMT_DXT1, FMT_DXT2_3, and FMT_DXT4_5 are stored in the L2 cache as 4x8 while their *_AS_16_16_16_16 counterparts are stored as 4x16."

Change 59115 on 2002/10/24 by jacarey@fl_jcarey2

RBBM Spec:
1. Add Extern_Trig_Cntl Register and Diagram
2. Update text for nqwait and gui_active related items.

Change 59100 on 2002/10/24 by ashishs@fl_ashishs_r400_win

update

Change 59060 on 2002/10/24 by tshah@fl_tshah

included description for CP_PIPE busy for Wait until condition. Hysteresis for GUI_ACTIVE

Change 59054 on 2002/10/24 by mkelly@fl_mkelly_r400_win_laptop

Update screen scissor test name in works..

Change 59053 on 2002/10/24 by mkelly@fl_mkelly_r400_win_laptop

SC Clip Rect tests, LINE_LOOP, STIPPLE, FSAA permutations...

Change 58970 on 2002/10/23 by ashishs@fl_ashishs_r400_win

Tests guard band clipping
4 primitives in 4 quadrants.
Each primitive has 6 vertices and first vert on vertical guard band.

Top Left Quadrant, gouraud shading
Top Right Quadrant, FLAT SHADING with START vertex as provoking vtx
Lower Right Quadrant, FLAT SHADING with END vertex as provoking vtx
Lower Left Quadrant, SIX TEXTURES, with COLOR0 as transparent

Method: 6 Vert Strip, first vert on vertical guard band
Expected Results: 4 primitives, four quadrants, no clipping, no discarding

Change 58948 on 2002/10/23 by rthambim@rthambim_r400_win_tor

Initial revision.

Change 58906 on 2002/10/23 by mzhu@mzhu_crayola_win_tor

Add in Colour Spaces.xls in perforce

Change 58841 on 2002/10/23 by frising@ma_frising

v.1.56
-fix up mapping of YCrCb to GPR in DATA_FORMAT table. Now Z(B)=Cb, Y(G)=Y, X(R)=Cr. (Cb are Cr were swapped before)
-cooorespondingly rename border colors to ACbYCr Black and ACbCrY Black. Note this is just a rename and does not affect actual values since Cr and Cb use the same value.

Change 58817 on 2002/10/23 by vgoel@fl_vgoel2

added vertex export bug report resulting from r400vgt_hos_PNT_adaptive

Change 58806 on 2002/10/23 by rbell@rbell_crayola_win_cvd

updated

Change 58801 on 2002/10/23 by jacarey@fl_jcarey2

Add ROP comment to GradFill Packet

Change 58743 on 2002/10/22 by frising@ma_frising

0.98
-added cube instruction implementation
-misc clean-up.

Change 58728 on 2002/10/22 by gregs@gregs_r400_win_marlboro

update

Change 58697 on 2002/10/22 by smoss@smoss_crayola_win

su tests

Change 58679 on 2002/10/22 by gregs@gregs_r400_win_marlboro

fixed Generic pads

Change 58630 on 2002/10/22 by rbell@rbell_crayola_win_cvd

Completed some more D2 tests

Change 58617 on 2002/10/22 by jacarey@fl_jcarey2

Update for Clearing the Register Update Flag

Change 58615 on 2002/10/22 by jacarey@fl_jcarey2

Document Updates for Incremental Register Updates

Change 58606 on 2002/10/22 by gregs@gregs_r400_win_marlboro

<added CONFIG_XSTRAP3 register and number of straps >

Change 58603 on 2002/10/22 by kmahler@kmahler_r400_doc_lib

Added More details and added Proposal for new PM4 Library and Real-Time support.

Change 58563 on 2002/10/22 by llefebvr@llefebvre_laptop_r400

Defined the memory exports better.

Change 58562 on 2002/10/22 by llefebvr@llefebvre_laptop_r400

Updated the SX interfaces.

Change 58525 on 2002/10/22 by jowang@jowang_R400_win

Modified M3 Test Plan after the review

Change 58522 on 2002/10/22 by georgev@ma_georgev

Updated to reflect changes and new stuff.

Change 58425 on 2002/10/21 by khabbari@khabbari_r400_win

added frame rate conv test list

Change 58419 on 2002/10/21 by frising@ma_frising

0.97
-initial check-in.

Change 58369 on 2002/10/21 by frising@ma_frising

v.1.55

- Added DEGAMMA_CNTL_ALL field to texture constant to indicate if degammed textures should be stored as 8-bit or 16-bit in TC. Noted that this field is not currently supported. Also noted what the default conversion are in lieu of this support along with bringing all the degamma documentation inline with the current degamma plans.

- Added REQUEST_SIZE field to texture and vertex constant to indicate if read requests should be 256 or 512 bit.

- Defined R400_TP_NAN to be 0x7FE00000 (used for out of range vertex fetches)

- *16_EXPAND is converted to 16.16 signed fixed point now.

- *MPEG is clamped to [-256..255] pre filter (was [-255..255])

Change 58350 on 2002/10/21 by rfevreau@rfevreau_r400_win

New goldens with PClk set to 165MHz
Test list update

Change 58329 on 2002/10/21 by mzhu@mzhu_crayola_win_tor

Update DxGRPH_PITCH and DxOVL_PITCH

Change 58321 on 2002/10/21 by jowang@jowang_R400_win

added M3 comments

Change 58318 on 2002/10/21 by mzhu@mzhu_crayola_win_tor

Update interface signals, DCP_DMIF_grph1_pitch, DCP_DMIF_grph2_pitch, DCP_DMIF_ovl1_pitch and DCP_DMIF_ovl2_pitch.

Change 58308 on 2002/10/21 by paulv@MA_PVELLA

Updated MHS portion of spec with latest and greatest features, info, etc.

Change 58307 on 2002/10/21 by csampayo@fl_csampayo_lt_r400

Cleaned up tests, added 1 new VGT test to test_list and updated the test tracker accordingly

Change 58257 on 2002/10/21 by khabbari@khabbari_r400_win

added frame rate conv

Change 58254 on 2002/10/21 by mzhu@mzhu_crayola_win_tor

Update 11.1.2 and 11.1.3 for D1GRPH and D2GRPH double buffer register
Add 11.20 for DCP CRC

Change 58253 on 2002/10/21 by mzhu@mzhu_crayola_win_tor

Add 8-bit 2101010 mode tests in 3.4.9.9

Change 58251 on 2002/10/21 by mzhu@mzhu_crayola_win_tor

Update DxGRPH double buffer register
Add DCP CRC register

Change 58213 on 2002/10/20 by gregs@laptop1

update

Change 58161 on 2002/10/19 by gregs@laptop1

initial device id numbers

Change 58120 on 2002/10/18 by csampayo@fl_csampayo_r400

Added bug# 88

Change 58095 on 2002/10/18 by vgoel@fl_vgoel2

updated for new bug

Change 58080 on 2002/10/18 by gregs@gregs_r400_win_marlboro

update

Change 58073 on 2002/10/18 by gregs@gregs_r400_win_marlboro

device_id update

Change 58032 on 2002/10/18 by jacarey@fl_jcarey2

Update diagram for clock gating

Change 58003 on 2002/10/18 by jasif@jasif_r400_win_tor

Updated test names and status.

Change 57994 on 2002/10/18 by ashishs@fl_ashishs_r400_win

updated a comment

Change 57991 on 2002/10/18 by jacarey@fl_jcarey2

Update documents for Incremental Register Update

Change 57976 on 2002/10/18 by ashishs@fl_ashishs_r400_win

Highlighted the bug "r400cl_bary_texture_08.cpp"

Change 57941 on 2002/10/18 by bbloemer@ma_bbloemer

Updated .doc file.

Change 57939 on 2002/10/18 by ashishs@fl_ashishs_r400_win

updated test_list and Tracker for the corresponding change in file name viz r400cl_gabnd_04 to r400cl_gband_06

Change 57935 on 2002/10/18 by ashishs@fl_ashishs_r400_win

Deleted test r400cl_gband_04 and renamed it to r400cl_gband_06. Also updated the Validation_Approach_plan.doc since it had some errors with tests name and numberings.

Change 57933 on 2002/10/18 by jacarey@fl_jcarey2

1. Remove legacy versions of packets that were struck through
2. Add Incremental Register Write Support

Change 57890 on 2002/10/18 by jacarey@fl_jcarey2

Added src_data_format to the alphablend packet.

Change 57866 on 2002/10/17 by grayc@grayc_r400_win

for safekeeping ... expanded on test description

Change 57848 on 2002/10/17 by csampayo@fl_csampayo_r400

Added bug# 78

Change 57834 on 2002/10/17 by gregs@gregs_r400_win_marlboro

added R300 power

Change 57824 on 2002/10/17 by gregs@gregs_r400_win_marlboro

scaled from R300 part - ready.

Change 57784 on 2002/10/17 by jacarey@fl_jcarey2

Refer to the PA register spec for the format of the destination clipping parameters in the PM4 spec.

Change 57667 on 2002/10/17 by gregs@gregs_r400_win_marlboro

update

Change 57661 on 2002/10/17 by gregs@gregs_r400_win_marlboro

added to the discription of ROM_CLK register

Change 57576 on 2002/10/16 by gregs@laptop1

work in progress

Change 57574 on 2002/10/16 by gregs@laptop1

work in progress

Change 57527 on 2002/10/16 by llefebvr@llefebvre_laptop_r400

Clarifications and minor updates. Version 2.08.

Change 57510 on 2002/10/16 by csampayo@fl_csampayo_r400

Added 1 new HOS with index reset test, updated test_list and test tracker

Change 57494 on 2002/10/16 by smoss@smoss_crayola_win

update

Change 57475 on 2002/10/16 by ashishs@fl_ashishs_r400_win

updated

Change 57471 on 2002/10/16 by rbell@rbell_crayola_win_cvd

Completed host arbitration tests.

Change 57470 on 2002/10/16 by jacarey@fl_jcarey2

Documentation Updates for Programming Max Count while processing.

Change 57459 on 2002/10/16 by mkelly@fl_mkelly_r400_win_laptop

Add r400sc_parameterized_line_list_01 to tracker...

Change 57435 on 2002/10/16 by ashishs@fl_ashishs_r400_win

update

Change 57394 on 2002/10/16 by mzhu@mzhu_crayola_win_tor

Add 8-bit 2101010 graphics format
Add PIX_TYPE for selecting color space conversion parameters

Change 57382 on 2002/10/16 by jacarey@fl_jcarey2

Update to transfifo clock

Change 57363 on 2002/10/16 by efong@efong_r400_win_tor_doc

Coverage Metrics Proposal and users guide

Change 57337 on 2002/10/16 by jacarey@fl_jcarey2

Update for sclk_reg

Change 57253 on 2002/10/15 by csampayo@fl_csampayo_r400

VGT housekeeping

Change 57252 on 2002/10/15 by csampayo@fl_csampayo_r400

Added bug# 77

Change 57247 on 2002/10/15 by csampayo@fl_csampayo_r400

Added 5 new VGT index reset tests and updated test_list and test tracker

Change 57202 on 2002/10/15 by csampayo@fl_csampayo_r400

Updated test_list and test tracker for the following VGT tests:
r400vgt_vtx_export_very_very_simple_01
r400vgt_vtx_export_very_very_simple_02
r400vgt_vtx_export_very_very_simple_03

Change 57196 on 2002/10/15 by jacarey@fl_jcarey2

Update for SCLK_REG

Change 57184 on 2002/10/15 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 57177 on 2002/10/15 by beiwang@bei_depot

Updated the MCCI block diagram with the latest AGP path

Change 57175 on 2002/10/15 by jacarey@fl_jcarey2

1. Add CP_PERFMON_CNTL to CP register set
2. Documentation Updates for Clock Gating

Change 57165 on 2002/10/15 by jowang@jowang_R400_win

updated test plan with M3 features for 20bpp and CRC

Change 57157 on 2002/10/15 by smorein@smorein_r400

updated with register names

Change 57146 on 2002/10/15 by kmahler@kmahler_r400_doc_lib

Added some minor details, but not sure what?

Change 57086 on 2002/10/15 by jhoule@MA_JHOULE

1.54:
Added MIP_PACKING bit allowing to disable packing of the mip tail.

Change 57052 on 2002/10/15 by jacarey@fl_jcarey2

Clarify the number of DWORDs for ALU and Texture constant updates.

Change 56993 on 2002/10/14 by csampayo@fl_csampayo_r400

Closed bugs# 63,71,74. Added bug# 76

Change 56977 on 2002/10/14 by ashishs@fl_ashishs_r400_win

update

Change 56944 on 2002/10/14 by jacarey@fl_jcarey2

Update Diagram for Dynamic Clocking in the CP

Change 56936 on 2002/10/14 by csampayo@fl_csampayo_r400

Added 3 new VGT tests, updated test_list and test tracker

Change 56928 on 2002/10/14 by mkelly@fl_mkelly_r400_win_laptop

update...

Change 56918 on 2002/10/14 by csampayo@fl_csampayo_lt_r400

Some VGT housekeeping

Change 56881 on 2002/10/14 by llefebvr@llefebvre_laptop_r400

Loops, jumps and calls are now using a 13 bit address which allows to jump and call and loop around any control flow addresses (does not requires to be even anymore).

Change 56872 on 2002/10/14 by lseiler@lseiler_r400_win_marlboro

Updated Zplane format to include MultiSample bit

Change 56865 on 2002/10/14 by vgoel@fl_vgoel2

added frame buffer dump mismatch issue for tiled on and off mode.

Change 56855 on 2002/10/14 by smoss@smoss_crayola_win

su tests

Change 56838 on 2002/10/14 by kcorrell@kcorrell_r400_docs_marlboro_nb

update of MH document up to chapter 13

Change 56659 on 2002/10/11 by jayw@MA_JAYW

old needs updating

Change 56612 on 2002/10/11 by jacarey@fl_jcarey2

Update for DP_SRC_SOURCE

Change 56604 on 2002/10/11 by llefebvr@llefebvre_laptop_r400

Revision 2.06 of the spec.

Change 56601 on 2002/10/11 by rbell@rbell_crayola_win_cvd

Completed b&w offset floating point tests

Change 56593 on 2002/10/11 by jacarey@fl_jcarey2

Make SQ_CP_*_eventid buses 5 bits to cover all defined events.

Change 56586 on 2002/10/11 by dwong@cndwong2

changed the real-time signal names wired into CP (from overlay)

bits 5 and 6 of connections to the CP_rts_discretes input bus are changed to DISPx_CP_flip_proceed

Change 56581 on 2002/10/11 by tshah@fl_tshah

description of RBBM_DEBUG register write - snoop in RBBM's BIF_PIPE

Change 56579 on 2002/10/11 by kmahler@kmahler_r400_doc_lib

Some updates to support Random Shader Generator.

Change 56574 on 2002/10/11 by kmahler@kmahler_r400_doc_lib

Some more updates; This document is now mostly ready for the initial review to allow the first phase of development to begin.

Change 56558 on 2002/10/11 by jacarey@fl_jcarey2

B4 and B9 Booleans need to be set for NextChar

Change 56544 on 2002/10/11 by jhoule@MA_JHOULE

0.9.15:
Various updates in descriptions.
Updated TP_TC interface to 9 bits of pitch everywhere.

Change 56495 on 2002/10/10 by csampayo@fl_csampayo_r400

Added bug# 74

Change 56475 on 2002/10/10 by mzhu@mzhu_crayola_win_tor

Fix LUT fill for Floating point 16161616 mode - (0x0000 - 0x3BFF)

Change 56461 on 2002/10/10 by nbarbier@nbarbier_r400_win_tor

Added CRTC1 & CRTC2 enable and freeze signals.
Added CRTC1 & CRTC2 interrupts.

Change 56403 on 2002/10/10 by jasif@jasif_r400_win_tor

Updated test names and status.

Change 56398 on 2002/10/10 by grayc@grayc_r400_win

updates

Change 56386 on 2002/10/10 by rherrick@ma_rherrick_crayola

Implement Mult-sample stimulus application by TC client...

Change 56354 on 2002/10/10 by gregs@gregs_r400_win_marlboro

added block_busy_extender

Change 56350 on 2002/10/10 by jacarey@fl_jcarey2

Update pseudocode for LCC packet as processed by Real-time

Change 56318 on 2002/10/10 by rfevreau@rfevreau_r400_win

Added a sleep to the job file and added new DVO testlist

Change 56314 on 2002/10/10 by jacarey@fl_jcarey2

Clarification on ME_INIT for Mask Bit 9

Change 56313 on 2002/10/10 by csampayo@fl_csampayo_r400

Adeed bug# 73

Change 56303 on 2002/10/10 by jacarey@fl_jcarey2

Added Default Reset Control to ME init packet (bit 9 of mask)

Change 56213 on 2002/10/09 by gregs@laptop1

update

Change 56212 on 2002/10/09 by csampayo@fl_csampayo_r400

Some VGT housekeeping

Change 56171 on 2002/10/09 by csampayo@fl_csampayo_r400

1. Updated tests r400vgt_index_min_max_01 and _02 for multi-context
2. Added new VGT tests r400vgt_index_min_max_03 and _04
3. Updated test_list and the test tracker

Change 56126 on 2002/10/09 by mpersaud@mpersaud_r400_win_tor

Bumped document revision

Change 56123 on 2002/10/09 by mpersaud@mpersaud_r400_win_tor

Added TV_DOUT_interlace_en and TV_DOUT_tvout_en to interface.

Change 56082 on 2002/10/09 by jacarey@fl_jcarey2

Added VIP_CP_eof_ack to list.

Change 56051 on 2002/10/09 by gregs@laptop1

pin_strap_buscfg is 2 bits widw now (was 3 bits)

Change 56048 on 2002/10/09 by jacarey@fl_jcarey2

Invert polarity of the MH_CP_writeclean signal

Change 56033 on 2002/10/09 by mkelly@fl_mkelly_r400_win_laptop

Packed color example usage for VFD

Change 56008 on 2002/10/09 by mkelly@fl_mkelly_r400_win_laptop

```
* Rasterize 256 triangles, 1 packet. Each triangle should hit 4 quads.
  The test moves the triangle 2 quads at a time in X and
  2 quads in increasing Y.
* Update test_list
* Update test documentation in test tracker
```

Change 55960 on 2002/10/08 by nbarbier@nbarbier_r400_win_tor

Added description of 2nd DVO CRC.
Added description of TMDS data synchronizer.

Change 55950 on 2002/10/08 by gregs@gregs_r400_win_marlboro

< DEVICE_ID - work in progress >

Change 55945 on 2002/10/08 by ashishs@fl_ashishs_r400_win

update

Change 55935 on 2002/10/08 by jacarey@fl_jcarey2

Update description of the B0 2D Boolean for monochrome sources.

Change 55922 on 2002/10/08 by jacarey@fl_jcarey2

Only Set B0 Boolean for solid brushes if SRC_TYPE != Mono

Change 55904 on 2002/10/08 by bryans@bryans_crayola_doc

Update schedule to minimize Linux usage after midnight

Change 55891 on 2002/10/08 by rherrick@ma_rherrick_crayola

Removing FETCH_SHADOW from queuemgr/checker... Implementing setting of SCLK period in parser... Changing IDLE to work on SCLK instead of MCLK... Beginning implementation Fetch Multisample in checker (wanted to regress and check this change in before heading forward from this point)...

Change 55869 on 2002/10/08 by smoss@smoss_crayola_win

su tests

Change 55848 on 2002/10/08 by mpersaud@mpersaud_r400_win_tor

Update port names to match verilog implementation

Change 55816 on 2002/10/08 by jacarey@fl_jcarey2

Update spec for dst clip parameters to be positive only.

Change 55795 on 2002/10/08 by smoss@smoss_crayola_win

su tests

Change 55763 on 2002/10/08 by grayc@grayc_r400_win

initial release

Change 55758 on 2002/10/08 by mkelly@fl_mkelly_r400_win_laptop

Add Clay's multi-chip tests to SC special cases documentation

Change 55731 on 2002/10/07 by csampayo@fl_csampayo_r400

Added bug# 72

Change 55727 on 2002/10/07 by csampayo@fl_csampayo_r400

Added VGT index size test and updated test_list and test tracker

Change 55696 on 2002/10/07 by gregs@gregs_r400_win_marlboro

AGP bug "AD_20 stitch" - fixed

Change 55679 on 2002/10/07 by jacarey@fl_jcarey2

Added write confirm function for DMA engine at end of table.

Change 55669 on 2002/10/07 by rbell@rbell_crayola_win_cvd

Completed half res tests

Change 55570 on 2002/10/07 by csampayo@fl_csampayo_lt_r400

Added bug# 71

Change 55555 on 2002/10/07 by chwang@chwang_doc_r400_win_cvd

Update.

Change 55544 on 2002/10/07 by imuskatb@imuskatb_r400_win_cnimuskatb

updated

Change 55527 on 2002/10/07 by smoss@smoss_crayola_win

SU tests

Change 55513 on 2002/10/06 by ashishs@fl_ashishs_r400_win

update

Change 55506 on 2002/10/06 by lkang@lkang_r400_win_tor

updates

Change 55494 on 2002/10/05 by gregs@laptop1

update - tests passed.

Change 55440 on 2002/10/04 by gregs@gregs_r400_win_marlboro

update

Change 55399 on 2002/10/04 by jacarey@fl_jcarey2

Miscellaneous Updates for New 2D Packets

Change 55398 on 2002/10/04 by jacarey@fl_jcarey2

1. Removed Width from compound indices
2. Added 2D_Tri_Strip compound indice
3. Misc. Comments.

Change 55342 on 2002/10/04 by gregs@gregs_r400_win_marlboro

update

Change 55332 on 2002/10/04 by llefebvr@llefebvr_r400

Small update regarding allocs.

Change 55327 on 2002/10/04 by alleng@alleng_r400_win_marlboro

Additional information regarding the color and depth cache tests...

Change 55302 on 2002/10/04 by jimmylau@jimmylau_r400_win_tor

Updates SCL-CRTC interface specs based on interface review on Sept 25, 02

Change 55285 on 2002/10/04 by mkelly@fl_mkelly_r400_win_laptop

Enter new SC perf test

Change 55209 on 2002/10/03 by csampayo@fl_csampayo_r400

Added bug# 70

Change 55208 on 2002/10/03 by gregs@laptop1

added issues/notes related to pad ring.

Change 55199 on 2002/10/03 by csampayo@fl_csampayo_r400

Added 4 new DMA swap tests and updated test_list and the test tracker

Change 55179 on 2002/10/03 by gregs@laptop1

added DVO power pads + added ESD pads (not distributed yet) + equlized sides to approximately the same size + clean-up

Change 55164 on 2002/10/03 by mzhu@mzhu_crayola_win_tor

Modify the 3rd milestone tests

Change 55163 on 2002/10/03 by jacarey@fl_jcarey2

Updates to Gradfill packet
Update to comment on setting the 2D B3 Boolean

Change 55162 on 2002/10/03 by jacarey@fl_jcarey2

Boolean B3 should be '0' for SRC_TYPE=0.
Mark Earl will check-in the emulator update.

Change 55136 on 2002/10/03 by mpersaud@mpersaud_r400_win_tor

Updated interface as per interface spec review on September 27.

Change 55094 on 2002/10/03 by kmahler@kmahler_r400_doc_lib

Another intermediate revision.... getting closer :)

Change 55083 on 2002/10/03 by smorein@smorein_r400

initial version of performance counter spec, block leads and arch need to view and edit

Change 55036 on 2002/10/03 by csampayo@fl_csampayo_r400

VGT housekeeping

Change 54988 on 2002/10/03 by rbell@rbell_crayola_win_cvd

Completed digital output tests and teapot test.

Change 54971 on 2002/10/03 by jacarey@fl_jcarey2

Minor updates to register default values and minimal power-up sequence.

Change 54899 on 2002/10/02 by gregs@gregs_r400_win_marlboro

open issues update

Change 54893 on 2002/10/02 by gregs@gregs_r400_win_marlboro

Fixed the PadLIst name :)

Change 54882 on 2002/10/02 by gregs@gregs_r400_win_marlboro

DEF generator pages cleaned-up.

Change 54810 on 2002/10/02 by csampayo@fl_csampayo_r400

Forgot to save previous updates

Change 54802 on 2002/10/02 by csampayo@fl_csampayo_r400

Added 3 new VGT tests with negative index offsets and updated test_list and the test tracker

Change 54725 on 2002/10/02 by kmahler@kmahler_r400_doc_lib

Another intermediate revision of the initial spec.

Change 54701 on 2002/10/01 by gregs@gregs_r400_win_marlboro

update (Tuesday)

Change 54679 on 2002/10/01 by tshah@fl_tshah

vgt skew count changes in decoument and code -- threshold must be a non-zero even number

Change 54634 on 2002/10/01 by smoss@smoss_crayola_win

SU tests

Change 54610 on 2002/10/01 by mzhu@mzhu_crayola_win_tor

Add LUT and frame buffer fill pattern for digital output

Change 54584 on 2002/10/01 by mkelly@fl_mkelly_r400_win_laptop

Update more stipple test documentation....

Change 54567 on 2002/10/01 by gregs@gregs_r400_win_marlboro

added mux for core_volt_cntl[1:0] pins

Change 54550 on 2002/10/01 by jacarey@fl_jcarey2

Partial updates for new 2D packets

Change 54523 on 2002/10/01 by mkelly@fl_mkelly_r400_win_laptop

Update new window offset tests.

Change 54518 on 2002/10/01 by mpersaud@mpersaud_r400_win_tor

Added to CRTC_TV.doc

Change 54490 on 2002/10/01 by jacarey@fl_jcarey_desktop

Miscellaneous clarifications to the RT event engine diagram.

Change 54476 on 2002/09/30 by kmahler@kmahler_r400_doc_lib

More updates to the initial design doc.

Change 54464 on 2002/09/30 by gregs@gregs_r400_win_marlboro

added DEBUG bus connections

Change 54447 on 2002/09/30 by gregs@gregs_r400_win_marlboro

closed ring with corners.

Change 54416 on 2002/09/30 by mdoggett@MA_MDOGGETT_LT

Added FetchMultiSample.

Change 54370 on 2002/09/30 by jacarey@fl_jcarey2

Added compound index details for Stretch Blit Support.

Change 54347 on 2002/09/30 by jacarey@fl_jcarey2

Add register to the poll_valid signal to align with data.

Change 54321 on 2002/09/30 by imuskatb@imuskatb_r400_win_cnimuskatb

updated

Change 54301 on 2002/09/30 by chwang@chwang_doc_r400_win_cvd

Update.

Change 54279 on 2002/09/30 by vgoel@fl_vgoel2

added bugzilla number 435

Change 54269 on 2002/09/30 by jcox@jcox_r400

fix size of chart

Change 54264 on 2002/09/30 by jcox@jcox_r400

Fix macro for dating chart

Change 54263 on 2002/09/30 by jcox@jcox_r400

Fix miscellaneous issues with Emulation regression summary and detail charts

Change 54261 on 2002/09/30 by mkelly@fl_mkelly_r400_win_laptop

Update usage of regress_r400

Change 54252 on 2002/09/29 by csampayo@fl_csampayo_lt_r400

Updated status for the VGT tests:
r400vgt_index_offset_04
r400vgt_index_offset_05

Change 54248 on 2002/09/29 by gregs@laptop1

Sunday update

Change 54184 on 2002/09/27 by rfevreau@rfevreau_r400_win

Added new job file and rg files for VIP full chip run
DVO Testlist update

Change 54165 on 2002/09/27 by jcox@jcox_r400

Add Emulator test plans to web

Change 54162 on 2002/09/27 by gregs@laptop1

Friday update

Change 54124 on 2002/09/27 by jcox@web_sync

Changes to publish charts to web (add close workbook macro)

Change 54120 on 2002/09/27 by gabarca@gabarca_crayola_win_cvd

updated, still not with the meeting conclussions

Change 54118 on 2002/09/27 by jcox@fl_jcox2_web

Add macros to output charts to web

Change 54069 on 2002/09/27 by csampayo@fl_csampayo_r400

Updated bug# 67

Change 54055 on 2002/09/27 by dougd@doug

Updated area of sq block based on netlist submitted 9/27/02

Change 54049 on 2002/09/27 by csampayo@fl_csampayo_r400

Some housekeeping...

Change 54045 on 2002/09/27 by kevino@kevino_r400_win_marlboro

latest

Change 54028 on 2002/09/27 by jacarey@fl_jcarey2

Clean Up Spec per actual connections.

Change 53988 on 2002/09/26 by gregs@gregs_r400_win_marlboro

update

Change 53953 on 2002/09/26 by smoss@smoss_crayola_win

SU tests

Change 53900 on 2002/09/26 by jacarey@fl_jcarey_desktop

Make 2D Default registers write only. Readable via the ME_STATMUX

Change 53857 on 2002/09/26 by jacarey@fl_jcarey_desktop

Update Event_Write packet to include all events

Change 53850 on 2002/09/26 by nbarbier@nbarbier_r400_win_tor

Updated interface document after review meeting.

Change 53842 on 2002/09/26 by rfevreau@rfevreau_r400_win

Docs updates

Change 53804 on 2002/09/26 by rbell@rbell_crayola_win_cvd

updated

Change 53802 on 2002/09/26 by vgoel@fl_vgoel2

updated for closed bugs

Change 53795 on 2002/09/26 by prunstad@prunstad_r400_win_marl

Updated mh area.

Change 53760 on 2002/09/26 by mkelly@fl_mkelly_r400_win_laptop

Close out two bugs

Change 53709 on 2002/09/25 by imuskatb@imuskatb_r400_win_cnimuskatb

1st draft of sample counter test

Change 53699 on 2002/09/25 by gregs@gregs_r400_win_marlboro

update

Change 53678 on 2002/09/25 by csampayo@fl_csampayo_r400

Closed bug# 66.

Change 53638 on 2002/09/25 by tien@ma_spinach

updated TP and TC areas

Change 53622 on 2002/09/25 by smoss@smoss_crayola_win

update

Change 53614 on 2002/09/25 by jasif@jasif_r400_win_tor

Updated regression test names.

Change 53607 on 2002/09/25 by mkelly@fl_mkelly_r400_win_laptop

Poly offset / scale tests

Change 53556 on 2002/09/25 by jacarey@fl_jcarey_desktop

Update CP's Buffer Sizes to a 2^20 DWORD limit.

Change 53534 on 2002/09/25 by csampayo@fl_csampayo_r400

Added Bugzilla report# to bug# 67

Change 53531 on 2002/09/25 by tshah@fl_tshah

Logic for Field Affecting Operation (FAO)

Change 53503 on 2002/09/25 by mkelly@fl_mkelly_r400_win_laptop

Log potential bug

Change 53502 on 2002/09/25 by mkelly@fl_mkelly_r400_win_laptop

Log potential bug

Change 53499 on 2002/09/25 by mkelly@fl_mkelly_r400_win_laptop

Poly Offset combinations...

Change 53496 on 2002/09/25 by mkelly@fl_mkelly_r400_win_laptop

Document r400sc_poly_offset_03 test

Change 53477 on 2002/09/24 by gregs@gregs_r400_win_marlboro

update

Change 53472 on 2002/09/24 by gregs@gregs_r400_win_marlboro

update

Change 53466 on 2002/09/24 by csampayo@fl_csampayo_r400

Try again

Change 53464 on 2002/09/24 by csampayo@fl_csampayo_r400

Added bug#67

Change 53454 on 2002/09/24 by efong@efong_r400_win_tor_doc

Initial Leda Proposal

Change 53451 on 2002/09/24 by csampayo@fl_csampayo_r400

Added new VGT test. Updated test_list and test tracker

Change 53423 on 2002/09/24 by csampayo@fl_csampayo_r400

Closed bug# 61 and added bug# 66

Change 53396 on 2002/09/24 by gregs@gregs_r400_win_marlboro

package.xls is now the official r400 spreadsheet.

Change 53395 on 2002/09/24 by gregs@gregs_r400_win_marlboro

< NEW - using rv350 pads now >

Change 53382 on 2002/09/24 by imuskatb@imuskatb_r400_win_cnimuskatb

first pass for M3 test

Change 53372 on 2002/09/24 by imuskatb@imuskatb_r400_win_cnimuskatb

updated test lib

Change 53368 on 2002/09/24 by khabbari@khabbari_r400_win

test plan for lb released

Change 53367 on 2002/09/24 by mkelly@fl_mkelly_r400_win_laptop

Log bug, rectangle list broken when CL clipping enabled and clipped

Change 53361 on 2002/09/24 by georgev@ma_georgev

Added PV to master document.

Change 53359 on 2002/09/24 by alleng@alleng_r400_win_marlboro

Initial checkin of Performance Verification test plan...

Change 53356 on 2002/09/24 by khabbari@khabbari_r400_win

added HTOTAL_BY_8 to the interface

Change 53347 on 2002/09/24 by frivas@FL_FRivas

Update to HOS test description.

Change 53280 on 2002/09/24 by jacarey@fl_jcarey_desktop

Add Fix2Flt_Reg PM4 Packet for Video Folks.
  PM4 Spec Update for Packet
  Add Opcode to pm4_it_opcodes
New Unit Test Included

Change 53266 on 2002/09/24 by efong@efong_r400_win_tor_doc

The Excel spreadsheet version of the R400 Regression milestones.doc

Change 53242 on 2002/09/24 by gabarca@gabarca_crayola_win_cvd

added blink_sequence1 test

Change 53200 on 2002/09/23 by csampayo@fl_csampayo_r400

Added new VGT index reset test. Updated test_list and test tracker

Change 53167 on 2002/09/23 by jacarey@fl_jcarey_desktop

1. Comment Only Correction to the Microcode
2. Update Brush_X and Brush_Y pointers in PM4 spec to be of the range 0.0 to 7.0

Change 53165 on 2002/09/23 by frising@ma_frising

v.1.53
Updates to FetchMultiSample:
-Added Point Filter to MSAA_FILTER_FUNCTION. This will be used to return a specific sample.
-OFFSET_Z will be used to specify which sample to return.
-For fetchmultisample constant, renamed OUT fileds to RESULT and COLOR0 fields to MSAA. Also swapped their locations. This may be slightly unexpected for SW but makes RESULT_FORMAT align with DATA_FORMAT which is more consistent for TC/TP. Note these fields may go under an additional name change once the new MSAA registers had been named...all in the name of consistency.

Change 53162 on 2002/09/23 by jacarey@fl_jcarey_desktop

Clarifications to LCC and Set_Constant Packet

Change 53111 on 2002/09/23 by tshah@fl_tshah

Only CPQ_DATA_SWAP bit controls the CP_PUSH_* data swap

Change 53072 on 2002/09/23 by efong@efong_r400_win_tor_doc

Updated

Change 53056 on 2002/09/23 by jacarey@fl_jcarey_desktop

Documented ROP3's that are supported in R400 for 2D.

Change 53020 on 2002/09/23 by imuskatb@imuskatb_r400_win_cnimuskatb

updated

Change 53019 on 2002/09/23 by mkelly@fl_mkelly_r400_win_laptop

Poly Offset, 8 MSAA, HOS

Change 53007 on 2002/09/23 by jacarey@fl_jcarey_desktop

1. Update for oper=clrrepack function for Dst_Type==2
2. Update sizes for physical memories.

Change 52992 on 2002/09/23 by mpersaud@mpersaud_r400_win_tor

Rev 0.7 Mahendra Persaud
Added TV_CRTC_framestart_freq[1:0] port.

Change 52951 on 2002/09/22 by rthambim@rthambim_r400_win_tor

Expanded the address width from MH to BIF.

Change 52950 on 2002/09/22 by rthambim@rthambim_r400_win_tor

Added status signals.

Change 52949 on 2002/09/22 by rthambim@rthambim_r400_win_tor

Included Multi-Function changes.

Change 52866 on 2002/09/20 by csampayo@fl_csampayo_r400

Added 2 new VGT tests and updated test_list and the test tracker accordingly

Change 52854 on 2002/09/20 by mzhu@mzhu_crayola_win_tor

Add test plan for 3rd Milestone features

Change 52843 on 2002/09/20 by gregs@gregs_r400_win_marlboro

Friday's update

Change 52804 on 2002/09/20 by mkelly@fl_mkelly_r400_win_laptop

Log potential texture / baryc bug

Change 52794 on 2002/09/20 by mkelly@fl_mkelly_r400_win_laptop

Review and update SC test validation tracking...

Change 52756 on 2002/09/20 by csampayo@fl_csampayo_lt_r400

Added bug# 63

Change 52730 on 2002/09/20 by rherrick@ma_rherrick_crayola

Implementing SWAP mode in parser through to client interface... Change all tests to include SWAP_NONE required field...

Also implementing parts of RANDOM mode on address transactions...

Change 52722 on 2002/09/20 by rbell@rbell_crayola_win_cvd

Updated

Change 52648 on 2002/09/19 by csampayo@fl_csampayo_r400

Added 1 new VGT event handling test and updated the test_list and test tracker accordingly

Change 52615 on 2002/09/19 by mzhu@mzhu_crayola_win_tor

Update test plan for third milestone

Change 52610 on 2002/09/19 by csampayo@fl_csampayo_lt_r400

Some VGT housekeeping

Change 52551 on 2002/09/19 by rramsey@RRAMSEY_P4_r400_win

Clean up RTS data in the SC
Change sc_walker so that hw_scissor is passed down instead of recalculated in the walker
Remove registry check for covered edge bias fix, it's always on now
Add covered edge bias fix to RTL (sc_pipe)
Clean up some compile warnings in the qpp
Add some documentation on line_stipple

Change 52531 on 2002/09/19 by gabarca@gabarca_crayola_win_cvd

updated

Change 52506 on 2002/09/19 by mkelly@fl_mkelly_r400_win_laptop

Cycle SC_SAMPLE_CNTL permutations with all primtypes
Cycle MSAA, 192 packets

Change 52487 on 2002/09/19 by imuskatb@imuskatb_r400_win_cnimuskatb

updated test list

Change 52482 on 2002/09/19 by mkelly@fl_mkelly_r400_win_laptop

Primitive should be gouraud shaded, it is flat shaded.
Related to PARAM_GEN = true, CENTERS_ONLY, and SAMPLE_CENTER
If PARAM_GEN = false, gouraud shading works as expected.

Change 52475 on 2002/09/19 by gregs@gregs_r400_win_marlboro

added deafult values for rom_pads_reg

Change 52449 on 2002/09/19 by kmahler@kmahler_r400_doc_lib

Added Primlib documents to doc_lib.

Change 52431 on 2002/09/18 by gregs@gregs_r400_win_marlboro

It passes read_write test.

Change 52430 on 2002/09/18 by gregs@gregs_r400_win_marlboro

added note about auto-increment of index register

Change 52429 on 2002/09/18 by csampayo@fl_csampayo_r400

Added 5 new VGT prim tests and updated test_list and test tracker accordingly

Change 52426 on 2002/09/18 by gregs@gregs_r400_win_marlboro

added ROM pads control register

Change 52411 on 2002/09/18 by bryans@bryans_crayola_doc

Update for 09/16

Change 52401 on 2002/09/18 by imuskatb@imuskatb_r400_win_cnimuskatb

updated test list and dates

Change 52388 on 2002/09/18 by gregs@gregs_r400_win_marlboro

fixed couple of bit fields XSTRAP2 register

Change 52382 on 2002/09/18 by mdoggett@MA_MDOGGETT_LT

Added DXN

Change 52363 on 2002/09/18 by mzhu@mzhu_crayola_win_tor

Add more description in 11.1.2 and 11.1.3 for double buffer in horizontal retrace

Change 52342 on 2002/09/18 by mkelly@fl_mkelly_r400_win_laptop

close bug

Change 52334 on 2002/09/18 by mpersaud@mpersaud_r400_win_tor

Added vsync and hsync going to dac capture model

Change 52324 on 2002/09/18 by jacarey@fl_jcarey_desktop

Intrinsic function for unpacking colors to 8888 format

Change 52300 on 2002/09/18 by mkelly@fl_mkelly_r400_win_laptop

Cycle primitive types through all MSAA and JSS modes.

Change 52295 on 2002/09/18 by nbarbier@nbarbier_r400_win_tor

Added DOUT_I2C tests for M3.

Change 52294 on 2002/09/18 by jhoule@MA_JHOULE

Adding it for Tien (he had problems)

Change 52268 on 2002/09/18 by csampayo@fl_csampayo_r400

Added bug# 61

Change 52265 on 2002/09/18 by jacarey@fl_jcarey_desktop

Update for Alpha Color Defaults for RGB565 brush color format.

Change 52257 on 2002/09/18 by rbell@rbell_crayola_win_cvd

updated

Change 52230 on 2002/09/18 by jacarey@fl_jcarey_desktop

Update to SRC terms for small_text and hostdata_blt* packets (i.e. 32-bit SRC_OFFSET)
Updated the fixed constants that are in the C7 vertex shader constant.

Change 52229 on 2002/09/18 by jacarey@fl_jcarey_desktop

Update for 32-bit SRC_Offset term for Small_Text and HostData_Blt* packets
Clarification for Foreground and Background Colors = Dst_Type format

Change 52226 on 2002/09/18 by efong@efong_r400_win_tor_doc

Added in a whole bunch of visio diagrams for some of the models

Change 52221 on 2002/09/18 by jacarey@fl_jcarey_desktop

Update CP_STQ_AVAIL register fields

Change 52204 on 2002/09/18 by mkelly@fl_mkelly_r400_win_laptop

RB asserts when polygon with MSAA = 0,1,2,3 followed
by point list MSAA = 5. If polygon MSAA = 5 or 7 the
test completes as expected.

$/r400/devel/test_list/src/chip/gfx/pa/bugs

make r400sc_sp_sample_cntl_08_bug.emu

```
Assertion failed: (coverage&errmask) == 0, file ../../../../emu_lib/model/gfx/rb
/rb_color_model.cpp, line 334
make[2]: [run_emu] Error 3 (ignored)
```

Change 52182 on 2002/09/17 by nbarbier@nbarbier_r400_win_tor

Updated DAC, TMDS & DVO sections for M3 requirements.

Change 52181 on 2002/09/17 by csampayo@fl_csampayo_r400

Housekeeping update

Change 52180 on 2002/09/17 by csampayo@fl_csampayo_r400

Added 4 new VGT prim tests and updated test_list and the test tracker accordingly.

Change 52165 on 2002/09/17 by sbagshaw@sbagshaw

modified to add additional details to one-shot mode
renamed all of the registers to have a DC_ prefix

Change 52045 on 2002/09/17 by kevino@kevino_r400_win_marlboro

tcfux for texture offsets (sample shift)

Change 52031 on 2002/09/17 by vgoel@fl_vgoel2

added bugs 385, 386

Change 52030 on 2002/09/17 by imuskatb@imuskatb_r400_win_cnimuskatb

update

Change 52025 on 2002/09/17 by smoss@smoss_crayola_win

bug in primlib

Change 52024 on 2002/09/17 by rfevreau@rfevreau_r400_win

Updates

Change 52020 on 2002/09/17 by chwang@chwang_doc_r400_win_cvd

Update.

Change 52018 on 2002/09/17 by csampayo@fl_csampayo_r400

Housekeeping update

Change 52009 on 2002/09/17 by mdoggett@MA_MDOGGETT_LT

Added L2 Block Offsets

Change 51990 on 2002/09/17 by frivas@FL_FRivas

Update to status of a HOS test.

Change 51904 on 2002/09/16 by sbagshaw@sbagshaw

R400 Test Debug Bus for Toronto Crayola blocks

Change 51897 on 2002/09/16 by csampayo@fl_csampayo_r400

Added 3 new VGT prim tests and updated test_list and the test tracker accordingly.

Change 51864 on 2002/09/16 by mpersaud@mpersaud_r400_win_tor

Added Q_DCCIF_W256ONLY and Q_DCCIF_WC_TIMEOUT[5:0] to interface

Change 51792 on 2002/09/16 by frising@ma_frising

v.1.52
-MSAA_NUM_SAMPLES now replaces ARBITRARY_FILTER in FetchMultiSample constant. Users will be expected to set ARBITRARY_FILTER via the instruction when/if needed. Use of constant value for this case will be undefined (and will likely result in MSAA_NUM_SAMPLES being used).

Change 51786 on 2002/09/16 by kmahler@kmahler_r400_doc_lib

Added description for using Control Flow Boolean Constants and Control Flow Loop Constants.

Change 51784 on 2002/09/16 by kmahler@kmahler_r400_doc_lib

Changed format of Fetch Instruction that only perform "Set" type operations by removing the destination operand and the 'base' source operand (the constant register operand).

Added the following syntax changes:

1) Predicate stack as a destination operand cannot be an export register, thus all predicate setting operation must have a GPR as the destination.

2) Predicate setting operations CANNOT specify clamp, negate and absolute modifiers.

3) Kill operations CANNOT specify CLAMP.

4) ALU instructions that export to Position, Color, or Fog cannot be predicated.

5) KILL, MOVA, and PRED_SET type instructions may not be coissued with each other.

Added restriction that the shader program cannot use the Address Register Relative Mode within an exec_end or a cexec_end low-level control flow instruction.

Change 51767 on 2002/09/16 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 51748 on 2002/09/16 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 51741 on 2002/09/16 by vgoel@fl_vgoel2

added potential bug with dumpview in displaying output of test with high tessellation level

Change 51672 on 2002/09/15 by gregs@laptop1

< added comments about TST_CG_test_sel signal >

Change 51561 on 2002/09/13 by gregs@gregs_r400_win_marlboro

Friday update

Change 51495 on 2002/09/13 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 51490 on 2002/09/13 by csampayo@fl_csampayo_lt_r400

Added to test_list and updated status for test:
r400vgt_multi_prim_reset_index_all_01

Change 51449 on 2002/09/13 by gregs@gregs_r400_win_marlboro

<minor updates>

Change 51438 on 2002/09/13 by jacarey@fl_jcarey_desktop

Adjust Registers for Visibility

Change 51430 on 2002/09/13 by vgoel@fl_vgoel2

updated bugs which are resolved

Change 51331 on 2002/09/13 by gregs@gregs_r400_win_marlboro

< SDI and SDP swapped >

Change 51314 on 2002/09/13 by jacarey@fl_jcarey_desktop

Removed ROP=0xCC note from PM4 spec for brush types 0x6 and 0x7

Change 51233 on 2002/09/12 by vgoel@fl_vgoel2

added bug 365, PA hanging if vertex export is enabled.

Change 51228 on 2002/09/12 by whui@whui_r400_win_tor

update idct_status

Change 51201 on 2002/09/12 by csampayo@fl_csampayo_r400

1. Added 5 new VGT provoking vtx tests
2. Updated test_list and test tracker

Change 51196 on 2002/09/12 by imuskatb@imuskatb_r400_win_cnimuskatb

updated test after picasso change

Change 51090 on 2002/09/12 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 51069 on 2002/09/12 by jacarey@fl_jcarey_desktop

Update for oper=gmcdecode: Brush Type 0x6 and 0x7 set bit 15 of result

Change 51068 on 2002/09/12 by jacarey@fl_jcarey_desktop

32x1 Brush Types are not converted to 32bpp by the CP.

Change 51044 on 2002/09/12 by rbell@rbell_crayola_win_cvd

updated

Change 51003 on 2002/09/12 by jowang@jowang_R400_win

add large v downscale case to debug

Change 51001 on 2002/09/12 by jowang@jowang_R400_win

added debug scaling tests

Change 50985 on 2002/09/12 by mzhu@mzhu_crayola_win_tor

Update STEREOSYNC algorithm

Change 50910 on 2002/09/11 by gregs@gregs_r400_win_marlboro

update

Change 50873 on 2002/09/11 by efong@efong_r400_win_tor_doc

Updated

Change 50836 on 2002/09/11 by kmahler@kmahler_r400_doc_lib

Made the following changes:

1) Removed ALL references of clauses.

2) Moved Texture Instrution definitions from chapter 3 into Appendix.

3) Removed remaining portion of Chapter 3, since the information was duplicated in Chapter 4. Chapter 4 is now chapter 3.

4) Made major revisions to Chapter 8 "Programming Syntax And Usage".

- Re-ordered sections to improve flow.
- Updated the low-level control flow syntax.
- Added details for the mid-level control flow syntax.
- Updated predication instruction syntax and description.
- Removed previous vector and previous scalar syntax from ALU source operands.
- Enhanced register indexing.
- Updated ALU instruction syntax to properly describe vector and scalar operations that do not conform to the general syntax; mova type, kill type, and predicate setting type operations.
- Corrected the usage of 'instruction' where 'operation' was meant in the description of ALU instructions.

5) Removed chapter 10 "Program text file format" since it was redundent information.

6) Made minor enhancements to descriptions.

7) Added Cross-References where appropriate and started using a standard for these using the form:

see section header_number:"header_text" on page #

This allows hardcopy readers the ability to easily locate a reference (see section 8.4.2.3.1:"Vector Operations" on page 38 for an example).

8) Modified the Appendix:

- Removed the following due to duplication:
  - Syntax Reference.
  - Control Flow from Instruction Reference.
- Moved BNF further down.
- Revised Instruction Reference.
- Updated examples to mid-level syntax.

Change 50802 on 2002/09/11 by jcox@jcox_r400_win

Moving to r400/web directory

Change 50800 on 2002/09/11 by jcox@jcox_r400_win

Initial Checkin of R400 Web

Change 50799 on 2002/09/11 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 50758 on 2002/09/11 by csampayo@fl_csampayo_r400

Some housekeeping updates

Change 50728 on 2002/09/11 by mkelly@fl_mkelly_r400_win_laptop

Logged sq dump bug for record keeping

Change 50722 on 2002/09/11 by rfevreau@rfevreau_r400_win

Test #7

Change 50707 on 2002/09/11 by mkelly@fl_mkelly_r400_win_laptop

Simple triangle, polymode back face tri fill

Change 50647 on 2002/09/10 by gregs@gregs_r400_win_marlboro

update (only 3 ERRORs in build_io.log)

Change 50632 on 2002/09/10 by smoss@smoss_crayola_win

bug fix

Change 50623 on 2002/09/10 by smoss@smoss_crayola_win

su tests

Change 50615 on 2002/09/10 by rbell@rbell_crayola_win_cvd

updated

Change 50605 on 2002/09/10 by subad@MA_SUBA

Updated Area numbers of TCD with the new code

Change 50578 on 2002/09/10 by jacarey@fl_jcarey_desktop

Update POLYLINE pseudocode.

Change 50566 on 2002/09/10 by grayc@grayc_r400_win

added minutes for last two meetings

Change 50539 on 2002/09/10 by mkelly@fl_mkelly_r400_win_laptop

Updates...

Change 50485 on 2002/09/10 by csampayo@fl_csampayo_r400

Format update

Change 50456 on 2002/09/10 by llefebvr@llefebvre_laptop_r400

New spin of the SQ spec including some interface changes and auto-counter architectural changes (for multipass pixel/vertex shaders).

Change 50441 on 2002/09/10 by jimmylau@jimmylau_r400_win_tor

Minor changes in CRTC testplan document

Change 50413 on 2002/09/10 by jacarey@fl_jcarey_desktop

Added Cache_Flush_and_Invalidate_TS to Event_TimeStamp_Write packet

Change 50403 on 2002/09/10 by jacarey@fl_jcarey_desktop

Add Result Reuse to Diagram

Change 50398 on 2002/09/10 by jacarey@fl_jcarey_desktop

Update Indice count for Polyline packet

Change 50347 on 2002/09/09 by csampayo@fl_csampayo_r400

1. Added SU multi-context test
2. Updated test_list and test tracker accordingly

Change 50340 on 2002/09/09 by gabarca@gabarca_crayola_win_cvd

updated after meeting

Change 50339 on 2002/09/09 by gregs@gregs_r400_win_marlboro

update

Change 50332 on 2002/09/09 by gabarca@gabarca_crayola_win_cvd

added timing for each VGA mode

Change 50326 on 2002/09/09 by nbarbier@nbarbier_r400_win_tor

Updated DVO specs.

Change 50296 on 2002/09/09 by hartogs@fl_hartogs

Added section on determining the required amount of input data based on the draw initiator and the grouper regsters.
Added section on sanity checks for programming the grouper in major mode 1.

Change 50249 on 2002/09/09 by vgoel@fl_vgoel2

closed bug issue for PNTQL

Change 50246 on 2002/09/09 by mzhu@mzhu_crayola_win_tor

Update ICON and CURSOR X2 magnification algorithm.

Change 50213 on 2002/09/09 by frising@ma_frising

v.1.51
-Add in 32-bit clear color to texture constant layout for FetchMultiSample.

This involved replacing CLAMP_Z (which did nothing for FetchMultiSample) with FILTER_FUNCTION. Note that FILTER_FUNCTION is 3 bits (from 6).

Drivers will be required to do a resolve pass when using fast color clear for multisample render targets > 32 bits prior to a multisampe fetch. Currently we only have one such format (4x16).

In the future (r[v]450?), it may be desireable to fetch individual samples into the shader pipe. This might be accomplished by a FILTER_FUNCTION opcode that usurps the clear color bits. In this case (low performance anyways) a fast color expand might always be required. But again, this has not yet be designed and is future looking...

Change 50206 on 2002/09/09 by smoss@smoss_crayola_win

su tests

Change 50185 on 2002/09/09 by imuskatb@imuskatb_r400_win_cnimuskatb

change test to crtc2
updated test list

Change 50184 on 2002/09/09 by imuskatb@imuskatb_r400_win_cnimuskatb

updated

Change 50171 on 2002/09/09 by efong@efong_r400_win_tor_doc

Updated

Change 50160 on 2002/09/09 by chwang@chwang_doc_r400_win_cvd

Update.

Change 50156 on 2002/09/09 by csampayo@fl_csampayo_r400

Updated status for bug# 48

Change 50147 on 2002/09/09 by mzhu@mzhu_crayola_win_tor

Update STEREOSYNC algorithm

Change 50142 on 2002/09/09 by grayc@grayc_r400_win

update to add cp functions for mti_pli.dll compile

Change 50127 on 2002/09/09 by efong@efong_r400_win_tor_doc

Updated

Change 50052 on 2002/09/06 by gregs@gregs_r400_win_marlboro

io_internal.v ready - memory connection missing.

Change 50032 on 2002/09/06 by kevino@kevino_r400_win_marlboro

Updated to give more3 detail about which tests need to be covered for which tex dimensions

Change 50014 on 2002/09/06 by mkelly@fl_mkelly_r400_win_laptop

Initial check of sc sample control for centers and centroids in the sc_sp

Change 50011 on 2002/09/06 by mzhu@mzhu_crayola_win_tor

Date: September 6, 2002
Add more description for LUT BLACK_OFFSET and WHITE_OFFSET in chapter 11.8
Change icon mix algorithm for data clamping in chapter 11.14
Change cursor mix algorithm for data clamping in chapter 11.16

Change 49997 on 2002/09/06 by mkelly@fl_mkelly_r400_win_laptop

Update new stipple tests

Change 49994 on 2002/09/06 by jacarey@fl_jcarey_desktop

Correct booleans again for transbitblt

Change 49991 on 2002/09/06 by jacarey@fl_jcarey_desktop

Update boolean settings for

Change 49986 on 2002/09/06 by gabarca@gabarca_crayola_win_cvd

updated after meeting

Change 49983 on 2002/09/06 by rbell@rbell_crayola_win_cvd

updated for review

Change 49982 on 2002/09/06 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 49952 on 2002/09/06 by rbell@rbell_crayola_win_cvd

updated list (all tests should be done now)!

Change 49943 on 2002/09/06 by vgoel@fl_vgoel2

entered bug 48

Change 49938 on 2002/09/06 by csampayo@fl_csampayo_r400

Updated Viz Query controls. Updated test status

Change 49903 on 2002/09/06 by mkelly@fl_mkelly_r400_win_laptop

Update with msaa tests from yesterday

Change 49888 on 2002/09/06 by jacarey@fl_jcarey_desktop

Document clearing of B4 and C2.x boolean if ROP[7:4] != ROP[3:0] for Blt and Text packets.

Change 49883 on 2002/09/06 by jacarey@fl_jcarey_desktop

Update for Timing

Change 49873 on 2002/09/05 by gabarca@gabarca_crayola_win_cvd

updated disp doc

Change 49860 on 2002/09/05 by csampayo@fl_csampayo_r400

Added bug# 47

Change 49850 on 2002/09/05 by vgoel@fl_vgoel2

added bugzill bug 337

Change 49832 on 2002/09/05 by csampayo@fl_csampayo_r400

Added and closed bug# 45

Change 49828 on 2002/09/05 by mpersaud@mpersaud_r400_win_tor

Updated interface and top level diagram net names to match code.

Change 49822 on 2002/09/05 by csampayo@fl_csampayo_r400

1. Added 1 new VGT test for event handling
2. Updated test_list and test tracker accordingly

Change 49810 on 2002/09/05 by frising@ma_frising

v.1.50
-Remove FMASK_BASE from FetchMultiSample constant. Larry tells me this is now encoded at a fixed offset from COLOR0_BASE.

Change 49802 on 2002/09/05 by frivas@FL_FRivas

Added a HOS Tri-Patch test.

Change 49786 on 2002/09/05 by askende@andi_r400_docs

new rev. 1.7

Change 49760 on 2002/09/05 by kcorrell@kcorrell_r400_docs_marlboro_nb

updated version - submitted now to update gart documentation

Change 49722 on 2002/09/05 by frivas@FL_FRivas

Added two HOS tests (Tri-Patch and Line-Patch).

Change 49717 on 2002/09/05 by llefebvr@llefebvre_laptop_r400

updated spec.

Change 49704 on 2002/09/05 by mkelly@fl_mkelly_r400_win_laptop

Log a local sc bug

Change 49698 on 2002/09/05 by bbuchner@fl_bbuchner_r400_win

updated spec to match changes made for TE timing

Change 49695 on 2002/09/05 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 49681 on 2002/09/05 by rbell@rbell_crayola_win_cvd

Added CSC tests, and updated LUT list (gamma 2.5 fill)

Change 49595 on 2002/09/04 by csampayo@fl_csampayo_r400

1. Added bug test case
2. Updated bug# 43 on Bug Tracker

Change 49594 on 2002/09/04 by vgoel@fl_vgoel2

added bug 32 to bugzilla (bug 333)

Change 49574 on 2002/09/04 by smoss@smoss_crayola_win

SU test

Change 49572 on 2002/09/04 by efong@efong_r400_win_tor_doc

Initial version

Change 49564 on 2002/09/04 by bryans@bryans_crayola_doc

Update for new regression times

Change 49562 on 2002/09/04 by bbuchner@fl_bbuchner_r400_win

removed unused interfaces.
fixed output unit data types for don't care data.

Change 49547 on 2002/09/04 by mkelly@fl_mkelly_r400_win_laptop

Update to be current with Bugzilla

Change 49546 on 2002/09/04 by rfevreau@rfevreau_r400_win

Minor fix

Change 49543 on 2002/09/04 by jowang@jowang_R400_win

updated with testcase names

Change 49525 on 2002/09/04 by mkelly@fl_mkelly_r400_win_laptop

Change test name, update tracker...

Change 49479 on 2002/09/04 by mpersaud@mpersaud_r400_win_tor

Updated interface and top level diagram to reflect removal of hdslewfilt and addition of syncgen.

Change 49477 on 2002/09/04 by jacarey@fl_jcarey2

Clarified Brush Expansion

Change 49462 on 2002/09/04 by bryans@bryans_crayola_doc

Update as per status meeting

Change 49417 on 2002/09/03 by gabarca@gabarca_crayola_win_cvd

Finished cleaning up the test plan and gave a name to each test

Change 49374 on 2002/09/03 by mmantor@mmantor_r400_win

added sc output of centers and xy data. Also passed the xy data on the ij bus between the SC and SQ

Change 49354 on 2002/09/03 by mkelly@fl_mkelly_r400_win_laptop

Initial check of MSAA centermost determination.

Change 49340 on 2002/09/03 by efong@efong_r400_win_tor_doc

Updated to the last review

Change 49336 on 2002/09/03 by bryans@bryans_crayola_doc

Update

Change 49332 on 2002/09/03 by vliu@vliu_r400_cnvliu100_win_cvd

Update

Change 49306 on 2002/09/03 by imuskatb@imuskatb_r400_win_cnimuskatb

updated

Change 49296 on 2002/09/03 by jhoule@MA_JHOULE

Bringing back the PostJune15 into official location.
Version is 1.49

Change 49290 on 2002/09/03 by jhoule@MA_JHOULE

Renamed SAMPLE_POSITION to SAMPLE_LOCATION in TFetchInstr.
Helps to reduce confusion with TP loops (esp. anisotropy walk taking point/bilinear/arbitrary samples at different positions).

Change 49285 on 2002/09/03 by gregs@gregs_r400_win_marlboro

Update ROM/CG/CGM/DBG

Change 49282 on 2002/09/03 by chwang@chwang_doc_r400_win_cvd

Update.

Change 49267 on 2002/09/03 by efong@efong_r400_win_tor_doc

Updated

Change 49244 on 2002/09/03 by llefebvr@llefebvre_laptop_r400

new spec

Change 49234 on 2002/09/02 by gabarca@gabarca_crayola_win_cvd

updated

Change 49232 on 2002/09/02 by askende@andi_r400_docs

fixed a typo on MUL_PREV2

Change 49141 on 2002/08/31 by askende@andi_r400_docs

new rev of the spec ...rev.1.6

Change 49139 on 2002/08/30 by gabarca@gabarca_crayola_win_cvd

updated test names

Change 49133 on 2002/08/30 by csampayo@fl_csampayo_r400

Revised tests requirements for VGT and SU

Change 49074 on 2002/08/30 by csampayo@fl_csampayo_r400

Closed bug# 24

Change 49072 on 2002/08/30 by rfevreau@rfevreau_r400_win

Updated to documentation

Change 49066 on 2002/08/30 by csampayo@fl_csampayo_r400

Updated status for the following tests:
r400vgt_suppress_eop_01
r400vgt_suppress_eop_02
r400vgt_suppress_eop_03
r400vgt_suppress_eop_04

Change 49055 on 2002/08/30 by khabbari@khabbari_r400_win

added start phase

Change 49040 on 2002/08/30 by grayc@grayc_r400_win

added initial notes for running a chip level graphic tests

Change 49017 on 2002/08/30 by jayw@MA_JAYW

added 3d addressing to tile address

Change 48997 on 2002/08/30 by mzhu@mzhu_r400_win_tor

Update chapter 10.6 - 10.12

Change 48971 on 2002/08/30 by mkelly@fl_mkelly_r400_win_laptop

Potential bug in line texturing/window scissor

Change 48962 on 2002/08/30 by jacarey@fl_jcarey_desktop

Removed StatMux Connections from Visio Diagram
The connections are listed in the CP Spec.

Change 48937 on 2002/08/29 by gabarca@gabarca_crayola_win_cvd

assigned test names except render graphics text display

Change 48903 on 2002/08/29 by khabbari@khabbari_r400_win

added urgency signal to lb/dcp interface

Change 48900 on 2002/08/29 by jacarey@fl_jcarey_desktop

Updates for controlling queued vs. non-queued transactions from micro engine.

Change 48890 on 2002/08/29 by jayw@MA_JAYW

first try at fragment surface addresses and formats

Change 48888 on 2002/08/29 by jacarey@fl_jcarey_desktop

Update sign extension note for the PLY_NEXTSCAN packet.

Change 48882 on 2002/08/29 by gregs@gregs_r400_win_marlboro

<pad-ring in progress >

Change 48874 on 2002/08/29 by lkang@lkang_r400_win_tor

update

Change 48865 on 2002/08/29 by mkelly@fl_mkelly_r400_win_laptop

    * Basic multi-texture tests
* 16 parameter, 64 dword texture test
* Expand VFD to handle up to 16 textures,
  see r400sc_tri_16_par_64_dwords_01.cpp for example useage

Change 48858 on 2002/08/29 by rbell@rbell_crayola_win_cvd

Finished more D2 tests

Change 48839 on 2002/08/29 by llefebvr@llefebvre_laptop_r400

Updated the SQ->SP interface. Added comment on the Constant load bus.

Change 48832 on 2002/08/29 by jacarey@fl_jcarey_desktop

Remove direct access to the Ib1 and Ib2 base/size fetchers

Change 48824 on 2002/08/29 by jacarey@fl_jcarey_desktop

Clarification for memory usage for Brush and Palette

Change 48814 on 2002/08/29 by mdoggett@MA_MDOGGETT_LT

Updated L2 formats table. Added L2 sector address maps. Started adding new L1 block offset calcs per format.

Change 48813 on 2002/08/29 by kevino@kevino_r400_win_marlboro

added rb_assert test case.

Change 48810 on 2002/08/29 by efong@efong_r400_win_tor_doc

Updated

Change 48805 on 2002/08/29 by frivas@FL_FRivas

Update to HOS section for new RECT patch tests.

Change 48786 on 2002/08/29 by jacarey@fl_jcarey2

Micro Engine Updates for NQ Flag

Change 48778 on 2002/08/29 by jacarey@fl_jcarey2

Add NQ Flag Processing....

Change 48768 on 2002/08/29 by csampayo@fl_csampayo_r400

Updated description for the following tests:
r400vgt_multi_context_04
r400vgt_multi_context_05
r400vgt_multi_context_06
r400vgt_multi_context_07
r400vgt_multi_context_08
r400vgt_multi_context_09
r400vgt_multi_context_10
r400vgt_multi_context_11

Change 48756 on 2002/08/29 by rbell@rbell_crayola_win_cvd

finished premultiplied alpha, some date changes

Change 48677 on 2002/08/28 by gabarca@gabarca_crayola_win_cvd

Updated the ATTR_PPAN =8, 8 dot case

Change 48669 on 2002/08/28 by jowang@jowang_R400_win

updated for tests and precision

Change 48668 on 2002/08/28 by jayw@MA_JAYW

frag in progress

Change 48649 on 2002/08/28 by vgoel@fl_vgoel2

entered non-reported bug reports to bugzilla and entered their bug number

Change 48638 on 2002/08/28 by vgoel@fl_vgoel2

added few more tests (hos and tone mapping, stereo vision)

Change 48623 on 2002/08/28 by jacarey@fl_jcarey2

Mono Brush Unpacking

Change 48596 on 2002/08/28 by khabbari@khabbari_r400_win

tvout spec for standard tv released

Change 48557 on 2002/08/28 by imuskatb@imuskatb_r400_win_cnimuskatb

fixed contorl signal crc calculation
updated quick_dacb test
added hpd to API
added hpd test

Change 48547 on 2002/08/28 by georgev@ma_georgev

Better support for floats and ints in range.

Change 48537 on 2002/08/28 by mkelly@fl_mkelly_r400_win_laptop

JSS 4x4 simple triangle

Change 48505 on 2002/08/28 by rbell@rbell_crayola_win_cvd

updates

Change 48487 on 2002/08/28 by ashishs@fl_ashishs_r400_win

CL tests: adding more UCP combo tests and updating tracker.

Change 48463 on 2002/08/28 by jacarey@fl_jcarey_desktop

Miscellaneous Spec Updates
Comments in Microcode for 2D Processing.

Change 48462 on 2002/08/28 by jacarey@fl_jcarey_desktop

Add B19 Boolean and Remove SRC_SC_BOTTOM_RIGHT_GMC default value

Change 48429 on 2002/08/27 by tshah@fl_tshah

performance counters added with document changes

Change 48421 on 2002/08/27 by ashishs@fl_ashishs_r400_win

opened bug r400cl_bary_texture_08_bug.cpp again and modified desciption.

Change 48419 on 2002/08/27 by vgoel@fl_vgoel2

updated bug report

Change 48414 on 2002/08/27 by ashishs@fl_ashishs_r400_win

deleted a redundant row related with the bug test r400cl_bary_texture_08_bug.cpp

Change 48409 on 2002/08/27 by ashishs@fl_ashishs_r400_win

deleted a redundant row related with the bug test r400cl_bary_texture_08_bug.cpp

Change 48406 on 2002/08/27 by gregs@gregs_r400_win_marlboro

Based onb RV350 design - R400 package definition (FC BGA pinout done (first revision)).

Change 48402 on 2002/08/27 by csampayo@fl_csampayo_r400

1. Updated test VFD for sourcing tex coord from vertex
2. Updated Bug Tracker accordingly

Change 48389 on 2002/08/27 by mzhu@mzhu_r400_win_tor

Update description for overlay gamma correction

Change 48371 on 2002/08/27 by grayc@grayc_r400_win

update for 8/23/2002 minutes

Change 48341 on 2002/08/27 by jacarey@fl_jcarey2

Final Update from Joe

Change 48327 on 2002/08/27 by ygiang@ygiang_r400_win_marlboro_p4

added: more sp tests

Change 48323 on 2002/08/27 by jacarey@fl_jcarey2

Update Foreground and Background Color Processing

Change 48322 on 2002/08/27 by ashishs@fl_ashishs_r400_win

update

Change 48290 on 2002/08/27 by jacarey@fl_jcarey2

Correction to CP_STAT

Change 48289 on 2002/08/27 by bryans@bryans_crayola_doc

Update status

Change 48284 on 2002/08/27 by gregs@gregs_r400_win_marlboro

< fixed PLL post divider control registers >

Change 48283 on 2002/08/27 by jacarey@fl_jcarey2

Update CP_STAT register bits

Change 48267 on 2002/08/27 by jacarey@fl_jcarey2

Add Boolean 19 (Misc Command)

Change 48266 on 2002/08/27 by jacarey@fl_jcarey2

More Updates

Change 48264 on 2002/08/27 by csampayo@fl_csampayo_r400

Bug test case

Change 48259 on 2002/08/27 by csampayo@fl_csampayo_r400

1. Added 1 new SU point sprite test

2. Updated test_list and test tracker accordingly

Change 48244 on 2002/08/27 by tshah@fl_tshah

rbbm interrupt logic and few changes in documentation

Change 48235 on 2002/08/27 by jimmylau@jimmylau_r400_win_tor

Update CRTC implementation specs :
update section on forcing H_COUNT & V_COUNT

Change 48226 on 2002/08/27 by bryans@bryans_crayola_doc

Update for arch/arch_<archname>.h change

Change 48223 on 2002/08/27 by jacarey@fl_jcarey2

Updated Small_Text Utility

Change 48218 on 2002/08/27 by mkelly@fl_mkelly_r400_win_laptop

Bug...

Change 48210 on 2002/08/27 by jacarey@fl_jcarey2

Utility to generate small_text characters.

Change 48187 on 2002/08/26 by georgev@ma_georgev

Added a few more comments.

Change 48184 on 2002/08/26 by gabarca@gabarca_crayola_win_cvd

added actual VESA mode test

Change 48183 on 2002/08/26 by gabarca@gabarca_crayola_win_cvd

detailed VESA modes

Change 48179 on 2002/08/26 by rbell@rbell_crayola_win_cvd

Updated schedule (lut bypass, d2 tests).

Change 48168 on 2002/08/26 by gregs@gregs_r400_win_marlboro

update ...

Change 48155 on 2002/08/26 by mmantor@mmantor_r400_win

updated 1st clk transfer of the sc to sq interface to define quad mask bits in the correct hardware/emulator order sp3 => sp0

Change 48144 on 2002/08/26 by csampayo@fl_csampayo_r400

Expanded format to include Bugzilla report# and, sorted by date.

Change 48123 on 2002/08/26 by efong@efong_r400_win_tor_doc

Updated with the chip level archs and configs

Change 48097 on 2002/08/26 by nbarbier@nbarbier_r400_win_tor

HPD Updates

Change 48096 on 2002/08/26 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 48092 on 2002/08/26 by llefebvr@llefebvre_laptop_r400

Added the new SQ->SP instruction interface.

Change 48070 on 2002/08/26 by efong@efong_r400_win_tor_doc

Updated

Change 48064 on 2002/08/26 by chwang@chwang_doc_r400_win_cvd

Update.

Change 48060 on 2002/08/26 by imuskatb@imuskatb_r400_win_cnimuskatb

updated

Change 48051 on 2002/08/26 by vliu@vliu_r400_cnvliu100_win_cvd

Update

Change 48043 on 2002/08/26 by rbell@rbell_crayola_win_cvd

updated

Change 48042 on 2002/08/26 by ashishs@fl_ashishs_r400_win

update

Change 48039 on 2002/08/26 by mkelly@fl_mkelly_r400_win_laptop

Memory crash when shader file doesn't exist..

Change 48035 on 2002/08/26 by rfevreau@rfevreau_r400_win

Update

Change 48028 on 2002/08/26 by tshah@fl_tshah

more description for rbbm_int_* registers for reset condition

Change 48021 on 2002/08/26 by mkelly@fl_mkelly_r400_win_laptop

Textured line where ST are exported directly to RGBA to visulize texture coordinates in color. Current there is a bug in the LLC quad pixel ST value when the primitive is not quad-aligned.

Change 48009 on 2002/08/26 by rfevreau@rfevreau_r400_win

Updates

Change 47998 on 2002/08/26 by kevino@kevino_r400_win_marlboro

Renamed TP to TPTC
changed "L2 filter no grouping" to "64 unique probe addresses"

Change 47985 on 2002/08/26 by efong@efong_r400_win_tor_doc

New version of it after review

Change 47982 on 2002/08/26 by efong@efong_r400_win_tor_doc

Deleted User Simulation Model Guide because renamed to VideoIp User Simulation Model Guide

Change 47981 on 2002/08/26 by efong@efong_r400_win_tor_doc

Changed name from the R400 User Simulation Model guide to VideoIP

Change 47968 on 2002/08/25 by gabarca@gabarca_crayola_win_cvd

updated list of tests

Change 47967 on 2002/08/25 by csampayo@fl_csampayo_r400

1. Added 1 new SU point sprite tests
2. Updated test_list and the test tracker accordingly

Change 47954 on 2002/08/25 by gabarca@gabarca_crayola_win_cvd

changed mono_reg_test name with mono_and_memory_mapped_reg_test

Change 47927 on 2002/08/23 by gabarca@gabarca_crayola_win_cvd

updated test plan

Change 47906 on 2002/08/23 by mzhu@mzhu_r400_win_tor

Add CRTC Event Trigger Signal Generation
Add Force V_COUNT, H_COUNT and field polarity now
Add Force H_COUNT now
Update StereoSync
Update digital TV timing
Re-order chapter 10.6 - 10.10

Change 47895 on 2002/08/23 by mkelly@fl_mkelly_r400_win_laptop

* Line list, start vertex at -8190 in HW space, clipped at -4096 Screen Space
* Line list, textured

Change 47874 on 2002/08/23 by frising@ma_frising

-remove statement about always using centriod sampling with jittered super-sampling. We may give meaning to both. No version bump.

Change 47867 on 2002/08/23 by frising@ma_frising

v.1.48
-Added SAMPLE_POSITION to texture instruction.

Specifies sampling position for gradient/LOD correction. Centroid of fragment should always be used with jittered super-sampling.
0=Centroid of fragment
1=Center of fragment

Change 47781 on 2002/08/23 by rbell@rbell_crayola_win_cvd

Some D2 updates

Change 47777 on 2002/08/23 by rbell@rbell_crayola_win_cvd

Completed B&W offset tests

Change 47775 on 2002/08/23 by kevino@kevino_r400_win_marlboro

Updates

Change 47774 on 2002/08/23 by omesh@ma_omesh

An upto date spreadsheet with accurate numbers showing RB directed tests status. This does not yet include tests for Gamma/Degamma and I also think Frank may need to update his section.

Change 47768 on 2002/08/23 by kevino@kevino_r400_win_marlboro

1st cut at TP verification xls

Change 47743 on 2002/08/23 by csampayo@fl_csampayo_r400

Update format

Change 47741 on 2002/08/23 by vgoel@fl_vgoel2

closed bug 6 and added bug 14

Change 47720 on 2002/08/23 by ygiang@ygiang_r400_win_marlboro_p4

update

Change 47717 on 2002/08/23 by mkelly@fl_mkelly_r400_win_laptop

Simple single line list

Change 47687 on 2002/08/22 by efong@efong_r400_win_tor_doc

Updated

Change 47632 on 2002/08/22 by vliu@vliu_r400_cnvliu100_win_cvd

Fixed colour space Excel sheet.

Change 47601 on 2002/08/22 by mzhu@mzhu_r400_win_tor

Add 64bpp digital output format

Change 47595 on 2002/08/22 by tshah@fl_tshah

typo fix for real time busy equation page-29

Change 47579 on 2002/08/22 by dglen@dglen_r400_dell

Adding new file

Change 47535 on 2002/08/22 by vgoel@fl_vgoel2

updated status of bug reports

Change 47522 on 2002/08/22 by csampayo@fl_csampayo_r400

1. Added 10 new VGT multi-context tests
2. Updated test_list and the test tracker accordingly

Change 47517 on 2002/08/22 by dglen@dglen_r400_dell

Added CRTC snapshot description

Change 47509 on 2002/08/22 by mpersaud@mpersaud_r400_win_tor

Removed some signals between dispout, crtc and the tvout.
Added interface with DCCG

Change 47491 on 2002/08/22 by mkelly@fl_mkelly_r400_win_laptop

Add two test descriptions, update comments in test

Change 47461 on 2002/08/21 by csampayo@fl_csampayo_r400

Updated for new vgt tests

Change 47454 on 2002/08/21 by ashishs@fl_ashishs_r400_win

CL test:r400cl_ucp_pointlist_01
64 point sprites with 6 textures and 6 ucp's enabled with point size set in PA_SU state register and also in PA_CL_POINT registers for clipping. Point Sprite UCP mode set to '3' viz. always expand and clip.

Change 47447 on 2002/08/21 by efong@efong_r400_win_tor_doc

updated

Change 47434 on 2002/08/21 by dglen@dglen_r400_dell

Change in section 7.7

Change 47424 on 2002/08/21 by rbell@rbell_crayola_win_cvd

Updated for blending tests

Change 47411 on 2002/08/21 by gregs@gregs_r400_win_marlboro

<display names changes update ++ >

Change 47344 on 2002/08/21 by vgoel@fl_vgoel2

added issue related with loading floating point texture image in local tone-mapping

Change 47336 on 2002/08/21 by askende@andi_r400_docs

modified the instruction interface from SQ to SP

Change 47324 on 2002/08/21 by vgoel@fl_vgoel2

vertex export bug related with number of position exports

Change 47305 on 2002/08/21 by jimmylau@jimmylau_r400_win_tor

Make changes to CRTC Controller Implementation specs
modify sections on : Generation of V_UPDATE
add sections on : CRTC_BLACK_COLOR
    disable CRTC in a safe place within the frame
    force VSYNC next line mode
    digital TV timings
    sample CRTC counts
    CRTC snapshots
    CRTC flow control

Change 47242 on 2002/08/21 by jacarey@fl_jcarey_desktop

Document Brush_Decode_Idle boolean

Change 47232 on 2002/08/21 by jacarey@fl_jcarey_desktop

1. Clarify clearing of 2D Allocation Flags
2. Added Brush_Decode_Busy boolean as bit 19...

Change 47229 on 2002/08/21 by vgoel@fl_vgoel2

added bug report for vertex export instruction em0

Change 47219 on 2002/08/21 by jacarey@fl_jcarey_desktop

Fix order of background and foreground colors

Change 47176 on 2002/08/20 by frising@ma_frising

v.1.47
-Make TRI_JUICE field 2 bits (like r350) with values: 0=0, 1=1/6, 2=1/4, 3=3/8. Also noted that trilinear filtering remains piecewise linear and continuous.
-Fix COLOR_FORMAT_OUT and COLOR_NUMBER_OUT fields in texture constant (used for fetchmulitsample) that were accidentally munged in a prior check-in.

Change 47174 on 2002/08/20 by dglen@dglen_r400_dell

Minor fixes to section 7

Change 47173 on 2002/08/20 by mzhu@mzhu_r400_win_tor

update DxGRPH register double buffer with STEREO_SELECT

Change 47131 on 2002/08/20 by frising@ma_frising

v.1.46
-Updated descriptions of get opcodes. Most notably specified that GetWeights now returns a horizontal and vertical lerp factor.
-Specified that the DATA_FORMAT field in texture constant can use any of the data formats in the DATA_FORMAT table while the cooresponding field in the vertex instruction can only use data formats valid for vertex data (also documented in DATA_FORMAT table).

Change 47120 on 2002/08/20 by mzhu@mzhu_r400_win_tor

Update test plan for third milestone

Change 47113 on 2002/08/20 by rbell@rbell_crayola_win_cvd

Completed alpha mode 2 tests

Change 47111 on 2002/08/20 by mpersaud@mpersaud_r400_win_tor

Updated interface diagram and added hd slew filter as its own block.

Change 47099 on 2002/08/20 by mzhu@mzhu_r400_win_tor

Add test plan for third milestone. Update black/white offset tests

Change 47065 on 2002/08/20 by mzhu@mzhu_r400_win_tor

Add LUT floating point data process, update DxGRPH register double buffer with STEREO_SELECT

Change 47061 on 2002/08/20 by dglen@dglen_r400_dell

Update to Display Outline for new CRTC features

Change 47035 on 2002/08/20 by rbell@rbell_crayola_win_cvd

Added totals, first D2 test is done.

Change 47021 on 2002/08/20 by vliu@vliu_r400_cnvliu100_win_cvd

Initial revision

Change 47007 on 2002/08/20 by vliu@vliu_r400_cnvliu100_win_cvd

Update

Change 47004 on 2002/08/20 by vliu@vliu_r400_cnvliu100_win_cvd

Initial revision

Change 46950 on 2002/08/20 by mdoggett@MA_MDOGGETT_LT

Updated DCC0 clients and GRB0 clients. Added separate VGA input to MHC.

Change 46945 on 2002/08/20 by jacarey@fl_jcarey2

Clarifications to Source and Destination formats for 2D.

Change 46924 on 2002/08/19 by gabarca@gabarca_crayola_win_cvd

updated registers

Change 46914 on 2002/08/19 by csampayo@fl_csampayo_r400

1. Added 2 new VGT test for multi context coverage
2. Updated test_list and the test tracker appropriately

Change 46911 on 2002/08/19 by nluu@nluu_r400_doclib_cnnb

- update forces

Change 46895 on 2002/08/19 by mkelly@fl_mkelly_r400_win_laptop

Primlib utility to check for double hit pixels in sc_quad_pair_proc_out.dmp

Concentric circle test to do a preliminary check on the new utility

Change 46891 on 2002/08/19 by lkang@lkang_r400_win_tor

updated for M2 & M3

Change 46883 on 2002/08/19 by nluu@nluu_r400_doclib_cnnb

- update forces

Change 46851 on 2002/08/19 by rbell@rbell_crayola_win_cvd

Added double-buffer tests

Change 46850 on 2002/08/19 by rfevreau@rfevreau_r400_win

Updated Status

Change 46847 on 2002/08/19 by csampayo@fl_csampayo_r400

Added instructions at the top

Change 46813 on 2002/08/19 by smoss@smoss_crayola_win

Bug

Change 46778 on 2002/08/19 by imuskatb@imuskatb_r400_win_cnimuskatb

updated

Change 46777 on 2002/08/19 by csampayo@fl_csampayo_r400

1. Added 4 new SU polymode degenerate tests
2. Updated test_list and test tracker accordingly

Change 46766 on 2002/08/19 by rbell@rbell_crayola_win_cvd

updated

Change 46761 on 2002/08/19 by efong@efong_r400_win_tor_doc

Updated

Change 46758 on 2002/08/19 by nluu@nluu_r400_doclib_cnnb

- update

Change 46751 on 2002/08/19 by ashishs@fl_ashishs_r400_win

corrected last submission

Change 46750 on 2002/08/19 by ashishs@fl_ashishs_r400_win

Hardware Accuracy related bug reported and documented.
(r400cl_bary_texture_08_bug.cpp, interpolator bug)

Change 46749 on 2002/08/19 by vliu@vliu_r400_cnvliu100_win_cvd

Update

Change 46735 on 2002/08/19 by jasif@jasif_r400_win_tor

Updated.

Change 46726 on 2002/08/19 by sbagshaw@sbagshaw

R400 Programming Guide for Toronto R400 Blocks
Kaleidoscope IP

Change 46724 on 2002/08/19 by chwang@chwang_doc_r400_win_cvd

Update.

Change 46695 on 2002/08/18 by csampayo@fl_csampayo_r400

1. Added 8 new SU polymode tests
2. Updated test_list and the test tracker accordingly

Change 46651 on 2002/08/16 by gabarca@gabarca_crayola_win_cvd

Updated interface with display

Change 46594 on 2002/08/16 by grayc@grayc_r400_win

doc to track minutes of validation meeting

Change 46565 on 2002/08/16 by peterp@MA_PETE_LT

Corrected SQ and TC "RV400 Qty" factors

Change 46560 on 2002/08/16 by rbell@rbell_crayola_win_cvd

updated

Change 46559 on 2002/08/16 by vgoel@fl_vgoel2

updated the "bug closed" and "change number" for bugs 4 & 5

Change 46554 on 2002/08/16 by vgoel@fl_vgoel2

added bug report for multi-context HOS test (bug 8)

Change 46547 on 2002/08/16 by jacarey@fl_jcarey2

Update for 2D Constants

Change 46536 on 2002/08/16 by jacarey@fl_jcarey2

Miscellaneous

Change 46530 on 2002/08/16 by vgoel@fl_vgoel2

added description for fix for bug 5 and 6 reported and added bug 7 description

Change 46513 on 2002/08/16 by rbell@rbell_crayola_win_cvd

updated

Change 46512 on 2002/08/16 by rbell@rbell_crayola_win_cvd

Updates

Change 46500 on 2002/08/16 by jacarey@fl_jcarey2

Updates for 2D Brush Decoding.

Change 46385 on 2002/08/15 by jowang@jowang_R400_win

changed coefficient range tests

Change 46324 on 2002/08/15 by vgoel@fl_vgoel2

added black pixels in white triangle bug

Change 46270 on 2002/08/15 by gregs@gregs_r400_win_marlboro

< changed direction of analog ports in io.v >

Change 46269 on 2002/08/15 by gregs@gregs_r400_win_marlboro

< added scratch register >

Change 46267 on 2002/08/15 by jacarey@fl_jcarey2

Update for Immediate Data Writes

Change 46260 on 2002/08/15 by csampayo@fl_csampayo_r400

Updated format

Change 46255 on 2002/08/15 by csampayo@fl_csampayo_r400

1. Added 3 new SU tests
2. Updated test_list and the test tracker accordingly

Change 46243 on 2002/08/15 by ygiang@ygiang_r400_win_marlboro_p4

update

Change 46239 on 2002/08/15 by vgoel@fl_vgoel2

reported two bugs

Change 46236 on 2002/08/15 by rbell@rbell_crayola_win_cvd

Updated schedule

Change 46206 on 2002/08/15 by jacarey@fl_jcarey2

Update Description of Vertex and Pixel Constants for 2D Processing.

Change 46187 on 2002/08/14 by georgev@ma_georgev

Updated spreadsheets.

Change 46183 on 2002/08/14 by gabarca@gabarca_crayola_win_cvd

Added handshake section

Change 46163 on 2002/08/14 by mzhu@mzhu_r400_win_tor

Update alpha blend and force graphics or overlay data to "0" for overlay-only or graphics-only case

Change 46138 on 2002/08/14 by llefebvr@llefebvre_laptop_r400

Changed the MASK mnemonics to KILL.
Added DST opcode.
Added MUL_PREV2 opcode.
Reordered the opcodes in primlib and SP.
Implemented the new KILL and SET SCALAR opcodes, they are now all comparing the ALPHA channel to 0.0f (instead of comapring against the RED channel).

Change 46126 on 2002/08/14 by efong@efong_r400_win_tor_doc

New document on chip integration tests

Change 46125 on 2002/08/14 by efong@efong_r400_win_tor_doc

Updated

Change 46111 on 2002/08/14 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 46099 on 2002/08/14 by askende@andi_r400_docs

new rev of the spec.

Change 46096 on 2002/08/14 by mpersaud@mpersaud_r400_win_tor

Updated specs to match Xilleon's tvout.

Change 46024 on 2002/08/14 by mzhu@mzhu_r400_win_tor

Update black/white offset

Change 46010 on 2002/08/14 by jacarey@fl_jcarey2

Update Setting 2D Boolean B0 if Brush_Type=0xF and SRC_TYPE = Color

Change 45994 on 2002/08/14 by jhoule@MA_JHOULE

TFetchInstr:
  Added SetGradientsH and SetGradientsV opcodes.
  Changed SetFilter4Weights opcode number.
  LOD_BIAS_{H|V} are now 7 bits, range [-4, 4).
  Added USE_REG_GRADIENTS.

TFetchConst:
  Changed description of TRI_JUICE.

Change 45907 on 2002/08/14 by bryans@bryans_crayola_doc

Updated...

Change 45889 on 2002/08/14 by jacarey@fl_jcarey2

Updates to Diagrams for Timing...

Change 45888 on 2002/08/14 by jacarey@fl_jcarey2

Update for Im_Load

Change 45863 on 2002/08/13 by gabarca@gabarca_crayola_win_cvd

put chapter 16 in a table

Change 45832 on 2002/08/13 by gabarca@gabarca_crayola_win_cvd

updated list of functional changes respect to previous chips

Change 45810 on 2002/08/13 by gabarca@gabarca_crayola_win_cvd

updated functional changes respect to previous chips

Change 45754 on 2002/08/13 by gregs@gregs_r400_win_marlboro

added examples of power state transitions programming via CP real time command streams + added description of spread spectrum circuits and registers.

Change 45743 on 2002/08/13 by jacarey@fl_jcarey2

Correct SRC_X calculation for HostData* Packets

Change 45680 on 2002/08/13 by gabarca@gabarca_crayola_win_cvd

Updated sec 16

Change 45674 on 2002/08/13 by gabarca@gabarca_crayola_win_cvd

Added section 16 Changes respect to previous ATI chips

Change 45673 on 2002/08/13 by mkelly@fl_mkelly_r400_win_laptop

Expand line width basic functionality...

Change 45657 on 2002/08/13 by csampayo@fl_csampayo_r400

1. Started log for bug #3
2. Expanded format to include accounting of bugs

Change 45615 on 2002/08/13 by jhoule@MA_JHOULE

Updated various TP/TC features.

Change 45512 on 2002/08/12 by askende@andi_r400_docs

new rev. 1.4

Change 45507 on 2002/08/12 by mzhu@mzhu_r400_win_tor

Update alpha blend mode 1 and 3

Change 45462 on 2002/08/12 by chwang@chwang_doc_r400_win_cvd

Update.

Change 45423 on 2002/08/12 by efong@efong_r400_win_tor_doc

Updated

Change 45420 on 2002/08/12 by chwang@chwang_doc_r400_win_cvd

Update.

Change 45414 on 2002/08/12 by csampayo@fl_csampayo_lt_r400

Updated status for the following tests:
r400su_point_sprite_01
r400su_point_sprite_02
r400su_point_sprite_03
r400su_point_sprite_04
r400su_point_sprite_05

r400su_point_sprite_06

Change 45405 on 2002/08/12 by mzhu@mzhu_r400_win_tor

Update for alpha blend key mode

Change 45398 on 2002/08/12 by imuskatb@imuskatb_r400_win_cnimuskatb

Updated DISPOUT
Added TAKEN to double buffer
M2 Dispout test
Updated Quick Emu

Change 45391 on 2002/08/12 by tshah@fl_tshah

typo - CP_RBBM_dma_busy is used for one of the wait until conditions

Change 45379 on 2002/08/12 by ashishs@fl_ashishs_r400_win

2nd bug tested and documented

Change 45364 on 2002/08/11 by ygiang@ygiang_r400_win_marlboro_p4

added: dot3 and dot4 shader tests

Change 45316 on 2002/08/09 by csampayo@fl_csampayo_r400

1. Updated image size for tests: r400su_point_sprite_01 and _02
2. Added 4 more SU point sprite tests
3. Updated test_list and test tracker accordingly

Change 45308 on 2002/08/09 by tho@tho_r400_win

new doc`

Change 45305 on 2002/08/09 by tho@tho_r400_win

run regress docs

Change 45292 on 2002/08/09 by whui@whui_r400_win_tor

update surface properties registers

Change 45262 on 2002/08/09 by georgev@ma_georgev

Updated.

Change 45232 on 2002/08/09 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 45220 on 2002/08/09 by mkelly@fl_mkelly_r400_win_laptop

Nan retain/kill on 2 vector exports from shader bug...

Change 45195 on 2002/08/09 by nbarbier@nbarbier_r400_win_tor

Initial Revision

Change 45180 on 2002/08/09 by bryans@bryans_crayola_doc

Update per last meeting

Change 45177 on 2002/08/09 by gregs@gregs_r400_win_marlboro

updated direct access to ROM pins (for writing).

Change 45156 on 2002/08/09 by dclifton@dclifton_r400

Updated zmin/zmax pinout info, updated area estimate

Change 45045 on 2002/08/08 by ashishs@fl_ashishs_r400_win

CL test:
A test to determine if the clip guard band works properly and that trivial reject works.
update to test tracker and test list.

Change 45021 on 2002/08/08 by nbarbier@nbarbier_r400_win_tor

Updated GENERICA & GENERICB inputs.

Change 45018 on 2002/08/08 by frivas@FL_FRivas

Update to HOS tests. Added edge detector test to "sheet 1" and accounted for it in the totals.

Change 44992 on 2002/08/08 by frivas@FL_FRivas

Updated status of different HOS tests.

Change 44907 on 2002/08/08 by csampayo@fl_csampayo_r400

1. Initial check in of Bug Tracker
2. Added bug test case

Change 44840 on 2002/08/08 by jacarey@fl_jcarey2

Correct note for vgt_busy status signal.

Change 44839 on 2002/08/08 by jacarey@fl_jcarey2

Remove 3D from the GradFill and AlphaBlend 2D packets.

Change 44771 on 2002/08/07 by jacarey@fl_jcarey2

Corrections to Packet Pseudocode.

Change 44759 on 2002/08/07 by dclifton@dclifton_r400

Updated for clipped polymode lines and points and updated point-line geometry drawing.

Change 44741 on 2002/08/07 by csampayo@fl_csampayo_r400

Updated status for the following tests:
r400su_point_sprite_01.cpp
r400su_point_sprite_02.cpp

Change 44684 on 2002/08/07 by lkang@lkang_r400_win_tor

2nd milestone review

Change 44680 on 2002/08/07 by imuskatb@imuskatb_r400_win_cnimuskatb

TMDS M2 Test
Updated Test list

Change 44665 on 2002/08/07 by jacarey@fl_jcarey2

1. Specify limit of ordinal #3 and #4 replications for the LCC packet.

Change 44663 on 2002/08/07 by grayc@grayc_r400_win

describes block owners/chip level verification

Change 44636 on 2002/08/07 by mkelly@fl_mkelly_r400_win_laptop

More JSS coverage...

Change 44537 on 2002/08/06 by gregs@gregs_r400_win_marlboro

spread spectrum

Change 44519 on 2002/08/06 by jacarey@fl_jcarey2

1. Document oper=gmcdecode in instruction table
2. Add 4 gprs to the micro engine

Change 44512 on 2002/08/06 by mkelly@fl_mkelly_r400_win_laptop

Textured pixel shader anti-aliased line example with VFD support.

Change 44469 on 2002/08/06 by jacarey@fl_jcarey2

Updated Interface from SQ for Events...4-bit event ID.
CP can ignore events that it does nothing for...

Change 44458 on 2002/08/06 by ashishs@fl_ashishs_r400_win

update

Change 44454 on 2002/08/06 by jasif@jasif_r400_win_tor

Updated.

Change 44416 on 2002/08/06 by tho@tho_r400_win

updated

Change 44406 on 2002/08/06 by vliu@vliu_r400_cnvliu100_win_cvd

Update

Change 44397 on 2002/08/06 by chwang@chwang_doc_r400_win_cvd

Update.

Change 44375 on 2002/08/06 by jacarey@fl_jcarey2

Update for oper=gmcdecode

Change 44372 on 2002/08/06 by efong@efong_r400_win_tor_doc

Updated

Change 44358 on 2002/08/06 by imuskatb@imuskatb_r400_win_cnimuskatb

Updated emulator to new crtc Block file
Updated Picasso to new register names
Updated Display Manager to handle TMDS stream
Completed Tmds stream library
Added one tmds test for M2

Change 44282 on 2002/08/05 by scroce@scroce_r400_win_marlboro

Update setup document

Change 44272 on 2002/08/05 by scroce@scroce_r400_win_marlboro

Some added info for the setup document

Change 44228 on 2002/08/05 by csampayo@fl_csampayo_r400

Added section 1.8

Change 44211 on 2002/08/05 by jacarey@fl_jcarey2

Update CP_STAT register and clarified that the CP_RBBM_rt_busy signal is not active if the RTEE is just performing polling operations.

Change 44192 on 2002/08/05 by wlawless@wlawless

new sample mask diagram

Change 44179 on 2002/08/05 by jacarey@fl_jcarey2

More Updates for Timing.

Change 44178 on 2002/08/05 by jcox@jcox_r400_win_orlando

added web macro

Change 44176 on 2002/08/05 by jacarey@fl_jcarey2

Shallow FIFOs Added Between PFP and Micro Engine for Timing..

Change 44096 on 2002/08/02 by mkelly@fl_mkelly_r400_win_laptop

* Implement vertex kill in the VFD for vertex buffer and constant
* Create a new class called PRIMITIVE_AA for AA dump file analysis
* Update tracker / test list

Change 44081 on 2002/08/02 by efong@efong_r400_win_tor_doc

updated

Change 44071 on 2002/08/02 by jayw@MA_JAYW

pre vacation check in.

Change 44063 on 2002/08/02 by mkelly@fl_mkelly_r400_win_laptop

Simple Poly offset check, needs SC fix to work...

Change 44053 on 2002/08/02 by jacarey@fl_jcarey2

Update 2D Spec for SRC_X pointer for small_text and host_data_blt
Most of microcode for Hostdata_Blt* packets...
Bitblt packets share subroutine with hostdata_blt packets...

Change 44048 on 2002/08/02 by gregs@gregs_r400_win_marlboro

<updates>

Change 44021 on 2002/08/02 by llefebvr@llefebvre_laptop_r400

New parameter generation scheme included in the spec.

Change 43966 on 2002/08/02 by mkelly@fl_mkelly_r400_win_laptop

Run line and point prims through JSS and MSAA...

Change 43913 on 2002/08/02 by jacarey@fl_jcarey2

Added oper=SIGNEXT for Micro Engine.

Change 43841 on 2002/08/01 by mzhu@mzhu_r400_win_tor

Update alpha fill pattern

Change 43815 on 2002/08/01 by mzhu@mzhu_r400_win_tor

Update LUT_INC, add UPDATE_TAKEN

Change 43814 on 2002/08/01 by scamlin@scamlin_crayola_win

orlando updates

Change 43779 on 2002/08/01 by jacarey@fl_jcarey2

Update bit widths of texture constants that are written
for Small_Text and HostData_Blt* packets.

Change 43775 on 2002/08/01 by mkelly@fl_mkelly_r400_win_laptop

Bresenham Control, converted R200 test...

Change 43773 on 2002/08/01 by jacarey@fl_jcarey2

Added Soft Reset control for the ROM block.

Change 43751 on 2002/08/01 by frivas@FL_FRivas

Update to HOS tests. Added complex model tests to tracker.

Change 43728 on 2002/08/01 by vgoel@fl_vgoel2

modified tracker to update hos status

Change 43710 on 2002/08/01 by mzhu@mzhu_r400_win_tor

Update for color cursor mode 2, LUT_BLACK_OFFSET/LUT_WHITE_OFFSET and LUT_INC implementation

Change 43706 on 2002/08/01 by csampayo@fl_csampayo_r400

Update

Change 43699 on 2002/08/01 by frivas@FL_FRivas

Added two HOS tests for precision.

Change 43687 on 2002/08/01 by ashishs@fl_ashishs_r400_win

update

Change 43685 on 2002/08/01 by rbagley@rbagley_ltxp

R400 Shader Programming Model

Checking in Laurent's revision of sequencer related updates. Also added initial specification of two language levels; this is out of date; updates in progress.

Change 43638 on 2002/08/01 by mkelly@fl_mkelly_r400_win_laptop

JSS 3x4 simple triangle

Change 43631 on 2002/08/01 by jacarey@fl_jcarey2

Update pseudocode for the ME_INIT packet.

Change 43571 on 2002/07/31 by csampayo@fl_csampayo_r400

Updates

Change 43543 on 2002/07/31 by gregs@gregs_r400_win_marlboro

<added IO_DC_xtalin signal>

Change 43509 on 2002/07/31 by jacarey@fl_jcarey2

Documentation Updates:
1. Viz Query
2. Memory Size Updates

3. Jump Address Added to Micro Instruction Format

Change 43496 on 2002/07/31 by rherrick@ma_rherrick_crayola

Implement absolute register directives for INFINITE control over specifying register offset values.

Change 43493 on 2002/07/31 by mkelly@fl_mkelly_r400_win_laptop

Testing lots of SC features with stipple enabled...

Change 43483 on 2002/07/31 by peterp@MA_PETE_LT

Corrected excel cell definition used by "Top" from "SQ" tab to remove !REF error.
Changed utilization factor to 70%.

Change 43423 on 2002/07/31 by jhoule@MA_JHOULE

Renamed TX_COORD_NORM to a more accurate TX_COORD_DENORM, keeping values intact, but sense more consistent.

Change 43417 on 2002/07/31 by frising@ma_frising

0.98 check-in

Change 43325 on 2002/07/31 by jacarey@fl_jcarey2

Add16 and Sub16 instructions sign-extend bit 13 (Used for 2D)
Min/Max and SCOMP assume that sign bit is bit 13 (Used for 2D)

Change 43240 on 2002/07/30 by semara@semara_r400_win_tor

add the smooth scrolling for the gfx

Change 43229 on 2002/07/30 by mkelly@fl_mkelly_r400_win_laptop

Line stipple variations...

Change 43220 on 2002/07/30 by jacarey@fl_jcarey2

Update Which Booleans are Set for 2D Packets with Source
Update Pitch = Pixels/32 in SRC/DST_PITCH_OFFSET values.

Change 43105 on 2002/07/30 by frivas@FL_FRivas

Update to HOS auto indexing tests.

Change 43095 on 2002/07/30 by smoss@smoss_crayola_win

SU tests

Change 43084 on 2002/07/30 by kcorrell@kcorrell_r400_docs_marlboro

updated area estimate for MH

Change 43077 on 2002/07/30 by efong@efong_r400_win_tor_doc

Updated

Change 43005 on 2002/07/29 by hartogs@fl_hartogs

Updated register addresses.

Change 42958 on 2002/07/29 by imuskatb@imuskatb_r400_win_cnimuskatb

Updated dc_crtc interface
Added some more M2 test to CRTC

Change 42949 on 2002/07/29 by mkelly@fl_mkelly_r400_win_laptop

128 packets, one triangle per packet, each hitting a subsample
in one JSS pixel. For each JSS_SAMPLE_SEL, each of 16 JSS pixels
are tested. JSS_SAMPLE_SEL is cycled from from 0 to 8. This test
ensures the tested JSS_SAMPLE_SEL is a unique value when compared
to all non-tested JSS_SAMPLE_SELs.

Change 42844 on 2002/07/29 by efong@efong_r400_win_tor_doc

updated

Change 42788 on 2002/07/26 by csampayo@fl_csampayo_r400

1. Added 2 new SU tests
2. Updated test_list and tracker, accordingly

Change 42781 on 2002/07/26 by fhsien@fhsien_r400_win_marlboro

Update Depth tests

Change 42737 on 2002/07/26 by scroce@scroce_r400_win_marlboro

Forgot to update TOC

Change 42735 on 2002/07/26 by scroce@scroce_r400_win_marlboro

Updating a bit to reflect changes in the release procedure and environment.

Change 42725 on 2002/07/26 by jasif@jasif_r400_win_tor

Updated after Testlist review meeting.

Change 42718 on 2002/07/26 by semara@semara_r400_win_tor

fix the index

Change 42654 on 2002/07/26 by jasif@jasif_r400_win_tor

Updated.

Change 42652 on 2002/07/26 by semara@semara_r400_win_tor

edit section 11 in the test plan
edit section 4 in the display interface

Change 42596 on 2002/07/26 by rbell@rbell_crayola_win_cvd

test list for ms2

Change 42530 on 2002/07/25 by ashishs@fl_ashishs_r400_win

CL TESTS: testing guard band clipping of edgeflags.

Change 42514 on 2002/07/25 by dougd@doug

updated area for SQ based on synthesis results

Change 42504 on 2002/07/25 by jasif@jasif_r400_win_tor

Finished graphics section.

Change 42499 on 2002/07/25 by mkelly@fl_mkelly_r400_win_laptop

Basic MSAA functionality tests...

Change 42442 on 2002/07/25 by gabarca@gabarca_crayola_win_cvd

improved description of LUT and fixed test case

Change 42367 on 2002/07/25 by mzhu@mzhu_r400_win_tor

Update DCP_tests_ms2.xls from Richard

Change 42362 on 2002/07/25 by mzhu@mzhu_r400_win_tor

Update graphics and overlay alpha blending

Change 42269 on 2002/07/24 by jasif@jasif_r400_win_tor

Fininshed hdp section and revised text section.

Change 42258 on 2002/07/24 by semara@semara_r400_win_tor

adding section 7, 8, and 9

Change 42234 on 2002/07/24 by mkelly@fl_mkelly_r400_win_laptop

* Update to JSS tests
* New MSAA test verifying all 8 subsamples in basic hit tests
* Update tracker
* Update SC test_list

Change 42233 on 2002/07/24 by nluu@nluu_r400_doclib_cnnb

- update specs for fast client protocol

Change 42140 on 2002/07/24 by imuskatb@imuskatb_r400_win_cnimuskatb

updated crtc.cpp to consider h_sync_start
fixed crtc_test.cpp
wrap around conter test in picasso
updated test list

Change 42100 on 2002/07/23 by frising@ma_frising

v.1.44
-It just occurred to me that having just one bit is insufficient for this. You really need 3 (one each for x,y,z), otherwise you'd get the wrong results if, for example, the user set WRAP_S to CLAMP and WRAP_T to CLAMP_TO_BORDER. I don't want to use three bits, so I've reworked things. The bit is now called NEAREST_CLAMP_POLICY and has the following behavior when nearest filtering:

Wrap/Repeat -> Wrap/Repeat
Mirror -> Mirror
Clamp to last texel -> Clamp to last texel
MirrorOnce to last texel -> MirrorOnce to last texel
Clamp half way to border color -> Clamp to last texel
MirrorOnce half way to border color -> MirrorOnce to last texel
Clamp to border color -> Clamp to border color
MirrorOnce to border color -> MirrorOnce to border color

This should allow D3D and OGL to set this bit 'once' and forget about it.

Change 42086 on 2002/07/23 by ashishs@fl_ashishs_r400_win

update

Change 42033 on 2002/07/23 by mkelly@fl_mkelly_r400_win_laptop

Update and corrected schedule test count versus sheet 1 counts.

Change 42000 on 2002/07/23 by gabarca@gabarca_crayola_win_cvd

added the VGA_CURSOR_BLINK_INVERT register

Change 41983 on 2002/07/23 by smoss@smoss_crayola_win

SU tests

Change 41973 on 2002/07/23 by jasif@jasif_r400_win_tor

Finished text mode testlist.

Change 41971 on 2002/07/23 by frising@ma_frising

v.1.43
-Add NEAREST_NO_BORDER bit to texture constant which controls if nearest filtering will every sample border color/texel. When this bit is asserted the mapping for the clamping policy when nearest filtering should be:
Wrap/Repeat -> Wrap/Repeat
Mirror -> Mirror
Clamp to last texel -> Clamp to last texel
MirrorOnce to last texel -> MirrorOnce to last texel
Clamp half way to border color -> Clamp to last texel
MirrorOnce half way to border color -> MirrorOnce to last texel
Clamp to border color -> Clamp to last texel
MirrorOnce to border color -> MirrorOnce to last texel

Change 41921 on 2002/07/23 by subad@MA_SUBA

updated area. Area decreased by re-structuring tx_fmt, oned and tiled muxes

Change 41905 on 2002/07/23 by ashishs@fl_ashishs_r400_win

update

Change 41898 on 2002/07/23 by smoss@smoss_crayola_win

shortened tests

Change 41883 on 2002/07/22 by gabarca@gabarca_crayola_win_cvd

Updated as per Joveria's test leist and last meetings

Change 41881 on 2002/07/22 by csampayo@fl_csampayo_r400

1. Added 2-D primitives to VGT section
2. Various format updates

Change 41867 on 2002/07/22 by gabarca@gabarca_crayola_win_cvd

updated

Change 41862 on 2002/07/22 by jasif@jasif_r400_win_tor

Updated.

Change 41822 on 2002/07/22 by mkelly@fl_mkelly_r400_win_laptop

* Basic JSS functional coverage
* Update test list
* Update tracker
* Added Plan vs. Actual graph to tracker sheet "schedule"

Change 41808 on 2002/07/22 by jasif@jasif_r400_win_tor

Updated.

Change 41784 on 2002/07/22 by gregs@gregs_r400_win_marlboro

<registers update>

Change 41781 on 2002/07/22 by frivas@FL_FRivas

Update. Changed name of a VGT HOS test to be completed.

Change 41778 on 2002/07/22 by vliu@vliu_r400_cnvliu100_win_cvd

Update

Change 41774 on 2002/07/22 by imuskatb@imuskatb_r400_win_cnimuskatb

Updated

Change 41756 on 2002/07/22 by chwang@chwang_doc_r400_win_cvd

Update.

Change 41743 on 2002/07/22 by tho@tho_r400_win

updated

Change 41742 on 2002/07/22 by tien@ma_spinach

updated TP/TC area estimates

Change 41731 on 2002/07/22 by efong@efong_r400_win_tor_doc

Updated

Change 41730 on 2002/07/22 by mkelly@fl_mkelly_r400_win_laptop

Improve test description and conveyance of test purpose in the image

Change 41729 on 2002/07/22 by efong@efong_r400_win_tor_doc

Updated

Change 41720 on 2002/07/22 by rbell@rbell_crayola_win_cvd

updated

Change 41717 on 2002/07/22 by georgev@ma_georgev

First revision from verification spec.

Change 41663 on 2002/07/19 by jacarey@fl_jcarey2

Fix Number of Bits Needed in the Brush, Palette, and Immed Base
Pointers.

Change 41654 on 2002/07/19 by gabarca@gabarca_crayola_win_cvd

updated

Change 41643 on 2002/07/19 by jasif@jasif_r400_win_tor

Updated.

Change 41620 on 2002/07/19 by mkelly@fl_mkelly_r400_win_laptop

1. Re-wrote JSS test to match latest SC spec.
2. Update test_list
3. Update tracker

Change 41600 on 2002/07/19 by llefebvr@llefebvre_laptop_r400

SQ backup.

Change 41597 on 2002/07/19 by jhoule@MA_JHOULE

Pre-SIGGRAPH submit (since it's exclusive open)

Change 41582 on 2002/07/19 by dclifton@dclifton_r400

Added description of rectangle primitive output. Updated state register description.

Change 41550 on 2002/07/19 by smoss@smoss_crayola_win

new x,y dim

Change 41531 on 2002/07/19 by bryans@bryans_crayola_doc

Update

Change 41524 on 2002/07/19 by bryans@bryans_crayola_doc

Minor changes to doc

Change 41450 on 2002/07/19 by jacarey@fl_jcarey2

Correct typo in diagram for write confirmation logic.

Change 41447 on 2002/07/19 by jacarey@fl_jcarey2

Write Confirmation Counter for CP/SC data coherency.

Change 41401 on 2002/07/18 by csampayo@fl_csampayo_r400

1. Added 4 new SU edgeflag tests
2. Updated framebuffer size for 2 tests
3. Updated test_list and test tracker accordingly

Change 41388 on 2002/07/18 by gabarca@gabarca_crayola_win_cvd

linked the test list excel into the "regression test table" icon at the end of the document

Change 41374 on 2002/07/18 by gabarca@gabarca_crayola_win_cvd

moved to the blocks/VGA directory

Change 41371 on 2002/07/18 by gabarca@gabarca_crayola_win_cvd

moved to the design/block/VGA directory

Change 41353 on 2002/07/18 by gabarca@gabarca_crayola_win_cvd

updated

Change 41352 on 2002/07/18 by jasif@jasif_r400_win_tor

Format change only.

Change 41332 on 2002/07/18 by jacarey@fl_jcarey2

Update Brush Address Range

Change 41330 on 2002/07/18 by jacarey@fl_jcarey2

Updates

Change 41321 on 2002/07/18 by jacarey@fl_jcarey2

Data written to Immd_Write_Confirm address generates a write confirmation.

Change 41308 on 2002/07/18 by bryans@bryans_crayola_doc

Update...

Change 41298 on 2002/07/18 by mzhu@mzhu_r400_win_tor

update for 2nd milestone part

Change 41296 on 2002/07/18 by gregs@gregs_r400_win_marlboro

<fixing previous mistake>

Change 41294 on 2002/07/18 by gregs@gregs_r400_win_marlboro

<moving from parts_lib to doc_lib>

Change 41293 on 2002/07/18 by georgev@ma_georgev

Updated for new tests.

Change 41287 on 2002/07/18 by gabarca@gabarca_crayola_win_cvd

updated

Change 41275 on 2002/07/18 by bryans@bryans_crayola_doc

Document that explains the new Chip Init library (usage and organization)

Change 41267 on 2002/07/18 by jacarey@fl_jcarey2

Clarification to LCC packet on preservation/setting of valid flag.

Change 41260 on 2002/07/18 by jacarey@fl_jcarey2

Update pseudocode for the issuance of event initiators to the scan converter for write confirm.

Change 41257 on 2002/07/18 by jayw@MA_JAYW

Thursday morning, minor updates

Change 41255 on 2002/07/18 by jasif@jasif_r400_win_tor

Updated.

Change 41225 on 2002/07/18 by jacarey@fl_jcarey2

Revert Last Change

Change 41213 on 2002/07/18 by jacarey@fl_jcarey2

1. Update LCC and Constant_Prefetch Packet formats
2. Constant write enables cleared for incremental update of constants.

Change 41208 on 2002/07/18 by fliljero@fl_frank

updated Load_Constant_Context write to BUFFER_BASE to match latest spec

Change 41183 on 2002/07/17 by csampayo@fl_csampayo_r400

1. Added 2 new tests to SU
2. Updated SU test_list
3. Updated test tracker accordingly

Change 41155 on 2002/07/17 by jacarey@fl_jcarey2

Updates for interpreting the 2D constants in the GMC.

Change 41148 on 2002/07/17 by jacarey@fl_jcarey2

Describe how the 2D Default Values are Applied by the CP.

Change 41112 on 2002/07/17 by smoss@smoss_crayola_win

SU Tests

Change 41107 on 2002/07/17 by jacarey@fl_jcarey2

1. Update Bit Width for Brush Address to HW support Logic
2. Revise 2D Default Registers

Change 41083 on 2002/07/17 by ashishs@fl_ashishs_r400_win

update

Change 41075 on 2002/07/17 by jowang@jowang_R400_win

more complete test plan

Change 41032 on 2002/07/17 by jayw@MA_JAYW

adding tile doc and minor depth address correction.

Change 40985 on 2002/07/17 by rbell@rbell_crayola_win_cvd

initial release

Change 40978 on 2002/07/17 by gabarca@gabarca_crayola_win_cvd

updated new fields

Change 40950 on 2002/07/16 by csampayo@fl_csampayo_r400

1. Added 2 new tests to SU
2. Updated test_list and test tracker

Change 40947 on 2002/07/16 by imuskatb@imuskatb_r400_win_cnimuskatb

Updated test list for M2

Change 40929 on 2002/07/16 by jacarey@fl_jcarey2

Update text for 2D/3D transition determination.

Change 40885 on 2002/07/16 by jayw@MA_JAYW

fixed error in tile address

Change 40881 on 2002/07/16 by mpersaud@mpersaud_r400_win_tor

Simplified write acknowledge return interface between dccif and client.

Change 40873 on 2002/07/16 by smoss@smoss_crayola_win

update

Change 40854 on 2002/07/16 by fhsien@fhsien_r400_win_marlboro

Update for Z tests

Change 40782 on 2002/07/15 by efong@efong_r400_win_tor_doc

Updated with M2 tests

Change 40771 on 2002/07/15 by csampayo@fl_csampayo_r400

1. Added SU test to validate provoking vertex
2. Updated SU's test_list accordingly
3. Updated the VGT test r400vgt_provoking_vtx_all_01
4. Updated test tracker with status for the test: r400su_provoking_vtx_rectangle_01

Change 40710 on 2002/07/15 by jayw@MA_JAYW

finished start of depth tile address calc

Change 40698 on 2002/07/15 by jacarey@fl_jcarey2

Update to ME_INIT for Header Dump Function

Change 40697 on 2002/07/15 by jasif@jasif_r400_win_tor

Updated.

Change 40693 on 2002/07/15 by tho@tho_r400_win

updated

Change 40691 on 2002/07/15 by llefebvr@llefebvre_laptop_r400

New sequencer spec.

Change 40666 on 2002/07/15 by jacarey@fl_jcarey2

Header Dump Can Be Used for Real-Time Streams

Change 40660 on 2002/07/15 by jacarey@fl_jcarey2

Update Header Dump Base Address Size

Change 40647 on 2002/07/15 by jasif@jasif_r400_win_tor

M2 VGA testlist.

Change 40632 on 2002/07/15 by jacarey@fl_jcarey2

Added a couple paragraphs to describe the Brush Decode support logic for the ME.

Change 40630 on 2002/07/15 by jacarey@fl_jcarey2

Added Local Addresses for Brush Decode Hardware

Change 40620 on 2002/07/15 by gregs@gregs_r400_linux_marlboro

<update>

Change 40609 on 2002/07/15 by jasif@jasif_r400_win_tor

Updated.

Change 40605 on 2002/07/15 by rbell@rbell_crayola_win_cvd

updated

Change 40600 on 2002/07/15 by jayw@MA_JAYW

Monday morning release.
start of cam and non-cam tags and
address generation

Change 40599 on 2002/07/15 by chwang@chwang_doc_r400_win_cvd

Update.

Change 40595 on 2002/07/15 by rbell@rbell_crayola_win_cvd

updated after test plan review

Change 40592 on 2002/07/15 by imuskatb@imuskatb_r400_win_cnimuskatb

Updated

Change 40589 on 2002/07/15 by tho@tho_r400_win

updated

Change 40582 on 2002/07/15 by tho@tho_r400_win

update

Change 40574 on 2002/07/15 by efong@efong_r400_win_tor_doc

Updated

Change 40571 on 2002/07/15 by nluu@nluu_r400_doclib_cnnb

- update

Change 40564 on 2002/07/15 by jacarey@fl_jcarey2

Updated Draw_Indx

Change 40535 on 2002/07/15 by jacarey@fl_jcarey2

ME_INIT side affect: Invalidates "valid flags" in the PFP.

Change 40473 on 2002/07/14 by vliu@vliu_r400_cnvliu100_win_cvd

Added conditional_write(...) function.

Change 40132 on 2002/07/12 by gabarca@gabarca_crayola_win_cvd

updated

Change 40121 on 2002/07/12 by jayw@MA_JAYW

Friday night checkin with template

Change 40120 on 2002/07/12 by fhsien@fhsien_r400_win_marlboro

Update for Z_functions

Change 40039 on 2002/07/12 by efong@efong_r400_win_tor_doc

Updated for VCD and FSDB and VPD support

Change 39993 on 2002/07/12 by frising@ma_frising

v.1.42
-allow TP to use FORMAT_COMP_* and related number format info for multisample fetches. Typically, the driver will want to make sure COLOR_FORMAT_OUT is consistent with these.

Change 39979 on 2002/07/12 by rbell@rbell_crayola_win_cvd

Secondary surface address test

Change 39977 on 2002/07/12 by rbell@rbell_crayola_win_cvd

updated for more milestone 2 and 3 tests

Change 39969 on 2002/07/12 by mzhu@mzhu_r400_win_tor

Update test plan for 2nd milestone

Change 39915 on 2002/07/12 by mkelly@fl_mkelly_r400_win_laptop

Update CL status

Change 39912 on 2002/07/12 by lkang@lkang_r400_win_tor

update

Change 39862 on 2002/07/11 by gabarca@gabarca_crayola_win_cvd

t

Change 39847 on 2002/07/11 by csampayo@fl_csampayo_lt_r400

Updated VGT section categories

Change 39830 on 2002/07/11 by csampayo@fl_csampayo_lt_r400

Updated the HOS Primitive Tessellation sub-section of the VGT section

Change 39819 on 2002/07/11 by frising@ma_frising

v.1.41
-multisample fetch basically done (note a few unused fields had to be moved in the tex constants)
-updated note on how degamma'd textures are stored in L2 (always 16 bit urf)

Change 39812 on 2002/07/11 by jacarey@fl_jcarey2

Real-Time Micro Engine does not have the direct write-back path as the Non-RT micro engine.

Change 39809 on 2002/07/11 by jacarey@fl_jcarey2

1. Boolean B0 is set for trans_bitblt packet
2. ME_INIT
  - Fixed typo of for vertex shader base size
  - Added note that pixel shader base must be greater than vertex
    shader base address.
3. Added notes to set B0 boolean for some of the new 2D packets.

Change 39764 on 2002/07/11 by georgev@ma_georgev

Fixed registers.

Change 39759 on 2002/07/11 by imuskatb@imuskatb_r400_win_cnimuskatb

Updated v1.2

Change 39739 on 2002/07/11 by mkelly@fl_mkelly_r400_win_laptop

Update showing priorities for Friday's meeting...

Change 39700 on 2002/07/11 by jacarey@fl_jcarey2

Fix Typos in Instr_Prefetch and Set_State Packets

Change 39679 on 2002/07/11 by jayw@MA_JAYW

0.1 rev mark

Change 39678 on 2002/07/11 by jayw@MA_JAYW

thursday update

Change 39662 on 2002/07/11 by rbell@rbell_crayola_win_cvd

updated schedule

Change 39637 on 2002/07/11 by smoss@smoss_crayola_win

account for differences in tests written and schedule

Change 39578 on 2002/07/10 by jacarey@fl_jcarey2

Miscellaneous

Change 39572 on 2002/07/10 by jimmylau@jimmylau_r400_win_tor

Update sections 5.6 and 5.7 after test plan review, to add more end cases and more detailed explanation to each test cases.

Change 39571 on 2002/07/10 by jacarey@fl_jcarey2

Clarifications for ALU and Texture write enables for LCC and
Set_Constant packets.

Change 39567 on 2002/07/10 by jacarey@fl_jcarey2

Miscellanesous Corrections for 2D Packets

Change 39565 on 2002/07/10 by csampayo@fl_csampayo_r400

Minor textual updates

Change 39563 on 2002/07/10 by csampayo@fl_csampayo_r400

Modified Provoking Vertex sub-section to include provoking vertex and edge flags

Change 39552 on 2002/07/10 by csampayo@fl_csampayo_r400

Added Rectangle List Processing sub-section to the SU section

Change 39486 on 2002/07/10 by khabbari@khabbari_r400_win

line buffer test plan for phase 2 released

Change 39399 on 2002/07/10 by jimmylau@jimmylau_r400_win_tor

Add section 10.4 in CRTC architecture and implementation specs, for double buffered registers
Add sections 5.6 to 5.8 in CRTC testplan, for overscan, interlaced timing mode and double buffered registers.

Change 39359 on 2002/07/10 by rbell@rbell_crayola_win_cvd

Updated for Kscope

Change 39350 on 2002/07/10 by rbell@rbell_crayola_win_cvd

Changed name of doc

Change 39347 on 2002/07/10 by smoss@smoss_crayola_win

SU tests

Change 39346 on 2002/07/10 by rbell@rbell_crayola_win_cvd

Updated file capture format, added optional macro signal capture

Change 39331 on 2002/07/10 by smoss@smoss_crayola_win

SU Tests

Change 39273 on 2002/07/09 by jacarey@fl_jcarey_desktop

Context is on data bits 12:10 !!!

Change 39245 on 2002/07/09 by jacarey@fl_jcarey_desktop

Micro Engine Updates

Change 39237 on 2002/07/09 by jacarey@fl_jcarey_desktop

1. Direct write path from micro engine behavior
2. Adjusted address of immd_swap in local address

Change 39220 on 2002/07/09 by csampayo@fl_csampayo_r400

1. Added 1 test to SU suite
2. Updated test_list accordingly
3. Updated relevant status on the test tracker

Change 39124 on 2002/07/09 by ygiang@ygiang_r400_win_marlboro_p4

added: shader test for debug

Change 39088 on 2002/07/09 by rbell@rbell_crayola_win_cvd

Updated status for LUT Host RW

Change 39078 on 2002/07/09 by csampayo@fl_csampayo_r400

Updated status for the following tests:
r400vgt_edgeflags_triangle_fan_01
r400vgt_edgeflags_triangle_wflags_01
r400vgt_edgeflags_triangle_strip_01
r400vgt_edgeflags_triangle_list_01
r400vgt_edgeflags_quad_strip_01
r400vgt_edgeflags_quad_list_01
r400vgt_edgeflags_polygon_01

Change 39062 on 2002/07/09 by mkelly@fl_mkelly_r400_win_laptop

VFD Edge flag support for Constant to EX1 operation with example test.

Change 39045 on 2002/07/09 by jayw@MA_JAYW

no change

Change 39016 on 2002/07/09 by mpersaud@mpersaud_r400_win_tor

Updated revision number

Change 39015 on 2002/07/09 by mpersaud@mpersaud_r400_win_tor

Tie off ICON_DCCARB_rswap & CURSOR_DCCARB_rswap to zero in DCCARB as it is not generated by either block

Change 39004 on 2002/07/09 by bryans@bryans_crayola_doc

There are 2 copies in the depot of this document: This one (version 4.2) is out of date. The most up to date one (version 4.4) resides in doc_lib/testenv. I left that copy there since it applies to both emulation and simulation.

Change 39003 on 2002/07/09 by bryans@bryans_crayola_doc

Updated as per status meeting

Change 38911 on 2002/07/08 by jacarey@fl_jcarey_desktop

1. Remove Set_Cf_Constant Packet
2. Added code for new Set_Constant packet
3. Preparations for writing constants to external memory

Change 38829 on 2002/07/08 by lseiler@lseiler_r400_win_marlboro

RB Arch spec: modified export buffer depth storage, added explanations

Change 38815 on 2002/07/08 by tho@tho_r400_win

updated

Change 38813 on 2002/07/08 by mmantor@mmantor_r400_win

assigned quad id to the lod correct term

Change 38785 on 2002/07/08 by csampayo@fl_csampayo_r400

Added status for the following SU tests:
r400su_provoking_vtx_point_01
r400su_provoking_vtx_line_01
r400su_provoking_vtx_triangle_01

Change 38743 on 2002/07/08 by mzhu@mzhu_r400_win_tor

Update for double buffer register

Change 38735 on 2002/07/08 by imuskatb@imuskatb_r400_win_cnimuskatb

updated (1st draft) of Dispout Test plan

Change 38722 on 2002/07/08 by rbell@rbell_crayola_win_cvd

updated test list after test plan review

Change 38697 on 2002/07/08 by imuskatb@imuskatb_r400_win_cnimuskatb

Updated

Change 38670 on 2002/07/08 by chwang@chwang_doc_r400_win_cvd

Update.

Change 38661 on 2002/07/08 by efong@efong_r400_win_tor_doc

Updated

Change 38658 on 2002/07/08 by jasif@jasif_r400_win_tor

Updated.

Change 38654 on 2002/07/08 by nluu@nluu_r400_doclib_cnnb

- update

Change 38652 on 2002/07/08 by tho@tho_r400_win

updated

Change 38648 on 2002/07/08 by vliu@vliu_r400_cnvliu100_win_cvd

Update

Change 38643 on 2002/07/08 by imuskatb@imuskatb_r400_win_cnimuskatb

Updated files for TMDS
Added new CMDKeys for DAC

Change 38642 on 2002/07/08 by efong@efong_r400_win_tor_doc

updated

Change 38618 on 2002/07/08 by rbell@rbell_crayola_win_cvd

updated

Change 38421 on 2002/07/05 by gabarca@gabarca_crayola_win_cvd

updated open items

Change 38412 on 2002/07/05 by nbarbier@nbarbier_r400_win_tor

Updated TMDS test descriptions

Change 38392 on 2002/07/05 by csampayo@fl_csampayo_lt_r400

Added Edge Flag Processing and Provoking Vertex Processing sub-sections (in the SU Section). Updated format of complete SU Section.

Change 38371 on 2002/07/05 by csampayo@fl_csampayo_r400

1. Added edge flag and provoking vertex sections to SU Polygon Mode 2. Moved r400su_edge_flag_01 to appropriate section

Change 38348 on 2002/07/05 by gabarca@gabarca_crayola_win_cvd

updated as pre meeting July 3

Change 38347 on 2002/07/05 by mkelly@fl_mkelly_r400_win_laptop

VFD edge flag support and example, note: waiting on bug fix in ccgen.cpp to be checked in.

Change 38344 on 2002/07/05 by gregs@gregs_r400_win_marlboro

<added indirect access to ROM memory

Change 38339 on 2002/07/05 by subad@MA_SUBA

updated area of tcd. Area increased by 2 sq.mm. bcos the correct area estimate of degamma logic was entered. Originally an estimate was entered.

Change 38338 on 2002/07/05 by jasif@jasif_r400_win_tor

Updated.

Change 38312 on 2002/07/05 by jacarey@fl_jcarey2

Clears 2D Flag when writing to the CONTEXT_ALLOCATION_FLAGS

Change 38294 on 2002/07/05 by jacarey@fl_jcarey2

New Addresses for Setting the Context Allocation Flags and the associated 2D flag.

Change 38285 on 2002/07/05 by jacarey@fl_jcarey2

Updates to ME_INIT

1. Instruction Thresholds need to be programmed for Real-Time
2. Side affects that occur when re-programming MAX_CONTEXT

Change 38275 on 2002/07/05 by tho@tho_r400_win

-updated test document
-updated model document

Change 38271 on 2002/07/05 by rbell@rbell_crayola_win_cvd

Added auto fill tests

Change 38257 on 2002/07/05 by bryans@bryans_crayola_doc

Add legacy version of display format (this needs to be updated)

Change 38254 on 2002/07/05 by bryans@bryans_crayola_doc

Update with misc. tasks

Change 38126 on 2002/07/04 by rbell@rbell_crayola_win_cvd

Added autofill test

Change 38125 on 2002/07/04 by mzhu@mzhu_r400_win_tor

Update for double buffer register

Change 38097 on 2002/07/04 by lkang@lkang_r400_win_tor

update

Change 38095 on 2002/07/04 by bryans@bryans_crayola_doc

Update for July 1 week

Change 38070 on 2002/07/04 by nbarbier@nbarbier_r400_win_tor

Updated Milestone 2 Tests

Change 37973 on 2002/07/03 by jasif@jasif_r400_win_tor

Updated.

Change 37951 on 2002/07/03 by llefebvr@llefebvre_laptop_r400

Backup

Change 37950 on 2002/07/03 by jasif@jasif_r400_win_tor

Updated.

Change 37926 on 2002/07/03 by jacarey@fl_jcarey2

Updates for Small_Text and HostData_Blt packets w.r.t. SRC_X term calculation.

Change 37925 on 2002/07/03 by mmantor@mmantor_r400_win

updated the sc_sq interface definition

Change 37905 on 2002/07/03 by rbell@rbell_crayola_win_cvd

Updated schedule, added more alpha blending tests

Change 37852 on 2002/07/03 by mkelly@fl_mkelly_r400_win_laptop

Good test for basic checks of sequence/scan conversion/shader operation...

Change 37847 on 2002/07/03 by rbell@rbell_crayola_win_cvd

Added zero expansion tests

Change 37832 on 2002/07/03 by mzhu@mzhu_r400_win_tor

Update for MH data return

Change 37831 on 2002/07/03 by hartogs@fl_hartogs

In description of register VGT_HOS_MAX_TESS_LEVEL, changed valid discrete levels from 1 thru 15 inclusive to 1 thu 14 inclusive.
This change was actually checked in by Brian Buchner yesterday, but he did not update the revision number or log.

Change 37798 on 2002/07/03 by imuskatb@imuskatb_r400_win_cnimuskatb

Updated

Change 37794 on 2002/07/03 by efong@efong_r400_win_tor_doc

updated

Change 37742 on 2002/07/02 by csampayo@fl_csampayo_r400

Updates:
1. Updated description/status for the following tests in the tracker:
r400vgt_provoking_vtx_all_01 r400vgt_hos_cubic_pos_pnt_discrete_01
2. Sorted test_list

Change 37706 on 2002/07/02 by gabarca@gabarca_crayola_win_cvd

Cleaned up descriptions of legacy non-standard VGA registers, added that DISP_START, BYTE PAN and ATTR_PPAN are loaded every line

Change 37705 on 2002/07/02 by tho@tho_r400_win

updated cursor test goldens
updated milestone 2 test list

Change 37702 on 2002/07/02 by jasif@jasif_r400_win_tor

Updated.

Change 37690 on 2002/07/02 by jhoule@MA_JHOULE

Changed format to removed level skipping.

Change 37681 on 2002/07/02 by bbuchner@fl_bbuchner_r400_win

tessellation engine creation

Change 37677 on 2002/07/02 by bbuchner@fl_bbuchner_r400_win

updated tess level constraints

Change 37675 on 2002/07/02 by vliu@vliu_r400_cnvliu100_win_cvd

Update

Change 37664 on 2002/07/02 by jacarey@fl_jcarey2

Added Microcode for 2D Paint Packet
Includes use of subroutines ... i.e. Call & Return Instructions.

Change 37642 on 2002/07/02 by jayw@MA_JAYW

initial rev

Change 37613 on 2002/07/02 by frivas@FL_FRivas

update to number of tests in "Schedule" sheet

Change 37607 on 2002/07/02 by gregs@gregs_r400_win_marlboro

added hardwired at the top of chip.v options (straps).

Change 37595 on 2002/07/02 by chwang@chwang_doc_r400_win_cvd

Update.

Change 37594 on 2002/07/02 by efong@efong_r400_win_tor_doc

Updated

Change 37563 on 2002/07/02 by jacarey@fl_jcarey2

Update for the arbitration in the RCIU

Change 37562 on 2002/07/02 by tho@tho_r400_win

updated

Change 37557 on 2002/07/02 by jasif@jasif_r400_win_tor

Updated.

Change 37549 on 2002/07/02 by rbell@rbell_crayola_win_cvd

updated for alpha blending

Change 37536 on 2002/07/02 by jacarey@fl_jcarey2

1. Update PFP Pseudocode for the case where the INDX_BASE/Size are not present in the draw_indx packet.
2. Updated algorithm for draw_indx in the PM4 Spec.

Change 37534 on 2002/07/02 by jacarey@fl_jcarey2

1. Update description for LCC packet.
2. Strike through packets that were obsolete/changed as of June 15th

Change 37477 on 2002/07/01 by csampayo@fl_csampayo_r400

1. Updated the display size, test description and other minor test format changes for the following tests:
r400vgt_dma_swap_indx16_01.cpp
r400vgt_dma_swap_indx32_01.cpp

2. Updated the tests descriptions on the test tracker for above tests

3. Deleted the following test from the test tracker and adjusted
Schedule accordingly:
r400vgt_hos_PNQ_lp_ln_cont_13_16_texture_lighting_projection.cpp

Change 37456 on 2002/07/01 by gregs@gregs_r400_linux_marlboro

<reset bring-up >

Change 37429 on 2002/07/01 by ashishs@fl_ashishs_r400_win

update

Change 37376 on 2002/07/01 by mkelly@fl_mkelly_r400_win_laptop

More tests validating reference vertex delta pixel calcs for baryc interp.

Change 37217 on 2002/06/28 by lkang@lkang_r400_win_tor

update

Change 37199 on 2002/06/28 by gabarca@gabarca_crayola_win_cvd

updated VGAHDP

Change 37174 on 2002/06/28 by csampayo@fl_csampayo_r400

Added the following new VGT tests and updated test tracker and test_list.

Change 37170 on 2002/06/28 by jasif@jasif_r400_win_tor

Updated.

Change 37101 on 2002/06/28 by jasif@jasif_r400_win_tor

Updated.

Change 36997 on 2002/06/28 by lkang@lkang_r400_win_tor

clock specs for bif and dc

Change 36991 on 2002/06/28 by gregs@gregs_r400_linux_marlboro

<type fix>

Change 36985 on 2002/06/28 by jacarey@fl_jcarey2

Add Defaults to the Spreadsheet

Change 36929 on 2002/06/27 by gregs@gregs_r400_linux_marlboro

< added IO_CG_refclk >

Change 36895 on 2002/06/27 by mzhu@mzhu_r400_win_tor

Update overlay zero expansion

Change 36880 on 2002/06/27 by jasif@jasif_r400_win_tor

Updated status. Reorganized features into more subsections.

Change 36872 on 2002/06/27 by gabarca@gabarca_crayola_win_cvd

Completed VGAHDP spec (except performance)

Change 36862 on 2002/06/27 by frising@ma_frising

Changes:
-Add INDEX_ROUND field to vfetch instruction (0=round, 1=truncate to negative infinity)
-Make OFFSET_X in vfetch instruction a signed 23-bit integer.

Change 36843 on 2002/06/27 by mpersaud@mpersaud_r400_win_tor

Updated documents to reflect actual DCCIF implementation

Change 36842 on 2002/06/27 by mpersaud@mpersaud_r400_win_tor

Cleared up operation of IDCT interface.
General document clean up and maintenance.

Change 36836 on 2002/06/27 by mkelly@fl_mkelly_r400_win_laptop

Perspective-Correct barycentric coordinate interpolation simple tests verifying combos of ref v0 locations.

Change 36831 on 2002/06/27 by jayw@MA_JAYW

pre-1st triange render check in

Change 36824 on 2002/06/27 by jacarey@fl_jcarey2

Document Polarity Inversion of the Deallocation FIFO Booleans
used by the Micro Engine.

Change 36802 on 2002/06/27 by mzhu@mzhu_r400_win_tor

Correct Y_alpha for graphics and overlay alpha blending

Change 36792 on 2002/06/27 by tho@tho_r400_win

emu feature list added

Change 36785 on 2002/06/27 by frising@ma_frising

Another 1.40 checkpoint.
-declare noise texture and related funcrionality as being unsupported on r400.
-adjust fetch opcodes for future growth. 0..15 = fetches, 16..23 = gets, 24..31 = sets
-move height specifier in SIZE field up 3-bits to allow for future growth of max 2D texture size.
-added DXN compressed 2 channel texture format for normals (and colors)
-specify that number format conversion will happen for compressed texture formats (e.g. DXT/DXN)
-make SIGNED_RF_MODEL_ALL apply to unsigned biased RF numbers. Evan pointed out that this is needed to do the *2-1 type mapping. Persumably since we have this for signed numbers this is straight forward to add.

-Until we understand what MS really wants, we are keeping our getgradients as is and not adding a setgradients/texldd.

Change 36784 on 2002/06/27 by gregs@gregs_r400_linux_marlboro

<remove ROM_strap_emu_desktop >

Change 36778 on 2002/06/27 by csampayo@fl_csampayo_r400

Updated status for the following VGT tests:

r400vgt_immed_index_tri_wflags_01
r400vgt_immed_index_tri_strip_01
r400vgt_immed_index_tri_list_01
r400vgt_immed_index_tri_fan_01
r400vgt_immed_index_rectangle_list_01
r400vgt_immed_index_polygon_01
r400vgt_immed_index_quad_strip_01
r400vgt_immed_index_quad_list_01

Change 36772 on 2002/06/27 by mzhu@mzhu_r400_win_tor

Correct REDD_LINE_COUNT logic for color cursor

Change 36755 on 2002/06/27 by jacarey@fl_jcarey2

Add Pixel and Vertex Shader Booleans and Status Signals.

Change 36747 on 2002/06/27 by jacarey@fl_jcarey2

Update Pseudocode for the POLYLINE PM4 Packet.

Change 36731 on 2002/06/27 by jacarey@fl_jcarey2

Add Vertex and Pixel Dealloc FIFO Full Signals to Boolean Register.

Change 36718 on 2002/06/27 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 36697 on 2002/06/27 by jacarey@fl_jcarey2

Miscellaneous Typos

Change 36614 on 2002/06/26 by csampayo@fl_csampayo_r400

Updated status for the following tests:
r400vgt_dma_index_quad_list_01
r400vgt_dma_index_quad_strip_01
r400vgt_dma_index_polygon_01

Change 36531 on 2002/06/26 by mzhu@mzhu_r400_win_tor

Add DMIF data return order

Change 36526 on 2002/06/26 by gregs@gregs_r400_win_marlboro

DRAM_select

Change 36521 on 2002/06/26 by mzhu@mzhu_r400_win_tor

Add more details for ICON and CURSOR implementation. Update LUT host read/write.

Change 36510 on 2002/06/26 by mkelly@fl_mkelly_r400_win_laptop

Update SC plan test approach for baryc interpolation reference pixel delta calculation.

Change 36508 on 2002/06/26 by gregs@gregs_r400_linux_marlboro

<resolving top level chip connections ... >

Change 36488 on 2002/06/26 by whui@whui_r400_win_tor

Test plan for R400 DCT

Change 36480 on 2002/06/26 by chwang@chwang_doc_r400_win_cvd

Rename.

Change 36479 on 2002/06/26 by chwang@chwang_doc_r400_win_cvd

Feature list for R400 IDCT.

Change 36476 on 2002/06/26 by rramsey@RRAMSEY_P4_r400_win

Clean up SC dumps
  Remove pa_sc.dmp since it is redundant
  Add sc_rbbm.dmp which only contains sc relevant reg writes so tb_sc runs faster
  Rearrange dump levels so only block level interfaces are dumped at level 1,
    hw accurate internals are dumped at level 2, and non-hw accurate are dumped
    at level 3
Update emu_dumps block diagram to reflect changes

Change 36473 on 2002/06/26 by imuskatb@imuskatb_r400_win_cnimuskatb

Updated

Change 36431 on 2002/06/26 by rherrick@ma_rherrick_crayola

Documentation for Multi-streamed RB Client interface...

Change 36394 on 2002/06/26 by jacarey@fl_jcarey2

Updates for Constant Write Enables Controlled by the LCC Packet
and not the Set_Constant packet.

Change 36302 on 2002/06/25 by csampayo@fl_csampayo_r400

Updated status for following tests:

r400vgt_dma_index_tri_list_01
r400vgt_dma_index_tri_fan_01
r400vgt_dma_index_tri_strip_01
r400vgt_dma_index_tri_wflags_01

Change 36274 on 2002/06/25 by ashishs@fl_ashishs_r400_win

update

Change 36258 on 2002/06/25 by lkang@lkang_r400_win_tor

updates

Change 36191 on 2002/06/25 by gregs@gregs_r400_linux_marlboro

<misc names changes >

Change 36171 on 2002/06/25 by jowang@jowang_R400_win

added SHIFT_IN_BLACK and BLACK_COLOR_*

Change 36163 on 2002/06/25 by dclifton@dclifton_r400

Identified gradients as subpixel gradients.

Change 36159 on 2002/06/25 by jacarey@fl_jcarey2

Update draw_indx pseudocode.

Change 36157 on 2002/06/25 by efong@efong_r400_win_tor_doc

Updated with autoreplication and autocentering for extended is done

Change 36154 on 2002/06/25 by jacarey@fl_jcarey2

Make microh to microm and mrh to mrm

Change 36145 on 2002/06/25 by csampayo@fl_csampayo_r400

Various updates to VGT section. Added test numbering to CL/VTE section

Change 36134 on 2002/06/25 by rfevreau@rfevreau_r400_win

Updated TMDS schedule

Change 36124 on 2002/06/25 by rbell@rbell_crayola_win_cvd

Added alpha blending tests, update schedule for this

Change 36091 on 2002/06/25 by jacarey@fl_jcarey2

Add note on management of the external memory surfaces for 2D.

Change 35994 on 2002/06/24 by csampayo@fl_csampayo_lt_r400

Update to fix counters for some tests in the CL/VTE section

Change 35974 on 2002/06/24 by jasif@jasif_r400_win_tor

Updated.

Change 35961 on 2002/06/24 by csampayo@fl_csampayo_lt_r400

Updated status for the following VGT tests:
r400vgt_dma_engine_09
r400vgt_dma_engine_10

Change 35947 on 2002/06/24 by gregs@gregs_r400_linux_marlboro

intel agp stp/busy protocol.

Change 35890 on 2002/06/24 by gregs@gregs_r400_linux_marlboro

fixing i/o

Change 35867 on 2002/06/24 by jacarey@fl_jcarey2

Status Spreadsheet Updated

Change 35862 on 2002/06/24 by rvelez@rvelez_r400_win_tor

Updated 6/24

Change 35830 on 2002/06/24 by frivas@FL_FRivas

Update. Finished HOS PNQ test.

Change 35826 on 2002/06/24 by jacarey@fl_jcarey2

Updates for Post-June15th Packet Changes

Change 35818 on 2002/06/24 by frivas@FL_FRivas

update to HOS PNQ test

Change 35817 on 2002/06/24 by efong@efong_r400_win_tor_doc

Updated so that 06/22/02 done on 06/30/02

Change 35814 on 2002/06/24 by efong@efong_r400_win_tor_doc

Updated

Change 35807 on 2002/06/24 by frivas@FL_FRivas

update to test counts in Schedule sheet

Change 35804 on 2002/06/24 by frivas@FL_FRivas

update to HOS PNQ test

Change 35799 on 2002/06/24 by chwang@chwang_doc_r400_win_cvd

Update.

Change 35795 on 2002/06/24 by vliu@vliu_r400_cnvliu100_win_cvd

Update

Change 35787 on 2002/06/24 by frivas@FL_FRivas

no change

Change 35784 on 2002/06/24 by tho@tho_r400_win

updated

Change 35773 on 2002/06/24 by rbell@rbell_crayola_win_cvd

Added overlay matrix transform tests

Change 35769 on 2002/06/24 by ctaylor@fl_ctaylor_r400_win_marlboro

Added Baryc description to help define Shader Pipe usage for pixel interpolation.

Change 35765 on 2002/06/24 by jacarey@fl_jcarey2

Updates for Microcode Width Increase to 74 bits

Change 35762 on 2002/06/24 by rbell@rbell_crayola_win_cvd

updated

Change 35746 on 2002/06/24 by ashishs@fl_ashishs_r400_win

update

Change 35743 on 2002/06/24 by frivas@FL_FRivas

Update

Change 35739 on 2002/06/24 by jacarey@fl_jcarey2

1. Microcode RAM is single-ported

Updated Sections:
6.11
7.28
7.29
7.13
7.14

Change 35732 on 2002/06/24 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 35641 on 2002/06/21 by ashishs@fl_ashishs_r400_win

update

Change 35639 on 2002/06/21 by csampayo@fl_csampayo_r400

Updated status for the following tests:
r400vgt_reuse_index_triangle_list_02
r400vgt_reuse_index_triangle_list_03

Change 35617 on 2002/06/21 by frivas@FL_FRivas

update to HOS PNT tests

Change 35604 on 2002/06/21 by jasif@jasif_r400_win_tor

Updated.

Change 35597 on 2002/06/21 by khabbari@khabbari_r400_win

auto coef cal released

Change 35571 on 2002/06/21 by jasif@jasif_r400_win_tor

Updated.

Change 35548 on 2002/06/21 by jacarey@fl_jcarey2

Add TC_RBBM_busy and add to RBBM_STATUS register
Change TP_RBBM_busy to TPC_RBBM_busy

Change 35546 on 2002/06/21 by jhoule@MA_JHOULE

Check in for Tien to look at a bit.

Change 35545 on 2002/06/21 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 35527 on 2002/06/21 by jacarey@fl_jcarey2

Clarify Event Initiators that generate signals from SQ and RC>

Change 35517 on 2002/06/21 by jacarey@fl_jcarey2

Update Documents

Change 35508 on 2002/06/21 by mkelly@fl_mkelly_r400_win_laptop

Simple CL test

Change 35501 on 2002/06/21 by rbell@rbell_crayola_win_cvd

Updated some graphics tests in testplan
Added two overlay basic tests (8888 and 2101010)

Change 35494 on 2002/06/21 by jacarey@fl_jcarey2

Fix Typo: T1 Constant should be T0 for Source informatoin.

Change 35352 on 2002/06/20 by jasif@jasif_r400_win_tor

Updated.

Change 35330 on 2002/06/20 by ashishs@fl_ashishs_r400_win

update

Change 35315 on 2002/06/20 by jasif@jasif_r400_win_tor

Updated.

Change 35282 on 2002/06/20 by jacarey@fl_jcarey2

1. Increased Depth of the Scratch Memory
2. Added Incremental_Update Boolean
3. Added Context_Dirty Boolean

Change 35264 on 2002/06/20 by fhsien@fhsien_r400_win_marlboro

Update errors

Change 35218 on 2002/06/20 by georgev@ma_georgev

Added changes from Lauraunt.

Change 35199 on 2002/06/20 by jasif@jasif_r400_win_tor

Updated.

Change 35176 on 2002/06/20 by jacarey@fl_jcarey2

Document that Type-0 packets cannot be used for instruction memory updates.

Change 35157 on 2002/06/20 by ashishs@fl_ashishs_r400_win

update

Change 35153 on 2002/06/20 by georgev@ma_georgev

Added new tests to list.

Change 35101 on 2002/06/20 by jacarey@fl_jcarey2

SRC1 into shifter is only 5 bits.

Change 35097 on 2002/06/20 by jacarey@fl_jcarey2

Fix Height and Width parameter sizes in packets

Change 35084 on 2002/06/19 by csampayo@fl_csampayo_r400

Updated status for the following tests:
r400vgt_dma_engine_01
r400vgt_dma_engine_02
r400vgt_dma_engine_03
r400vgt_dma_engine_04
r400vgt_dma_engine_05
r400vgt_dma_engine_06
r400vgt_dma_engine_07
r400vgt_dma_engine_08

Change 35082 on 2002/06/19 by mpersaud@mpersaud_r400_win_tor

Updated spec to be inline with actual implementation

Change 35053 on 2002/06/19 by rvelez@rvelez_r400_win_tor

Updated 6/19/02

Change 35048 on 2002/06/19 by lkang@lkang_r400_win_tor

updated emulation in vga, scaler, crtc, lb, & dispout.

Change 35028 on 2002/06/19 by jowang@jowang_R400_win

updated

Change 35008 on 2002/06/19 by jacarey@fl_jcarey2

Updates for IM_Load_Immediate Support

Change 35007 on 2002/06/19 by tho@tho_r400_win

-2x magnify cursor complete
-color cursor complete

Change 34985 on 2002/06/19 by jacarey@fl_jcarey2

1. Preparations for Context Management
2. Event Timestamp Discards Data If Event ID Does Not Match
3. Added DMA_Data_Source and Context_Dirty local Addresses.

Change 34977 on 2002/06/19 by jacarey@fl_jcarey2

Additions to Micro Engine Address Map
1. DMA_Data_Source
2. Context_Dirty Boolean

Change 34959 on 2002/06/19 by jacarey@fl_jcarey2

Update Base Addresses for Determining Incremental Updates.

Change 34953 on 2002/06/19 by mkelly@fl_mkelly_r400_win_laptop

Update to include control of Z_WRITE_ENABLE and ZFUNC... awaiting support updates to RB emu_lib...

Change 34936 on 2002/06/19 by imuskatb@imuskatb_r400_win_cnimuskatb

Updated

Change 34920 on 2002/06/19 by jowang@jowang_R400_win

updated version

Change 34918 on 2002/06/19 by frising@ma_frising

Another 1.40 chechpoint.
-go only with YUV replication (remove YUV interpolation)
-rough in DXN1 and DXN2 compressed normal formats

Change 34914 on 2002/06/19 by bryans@bryans_crayola_doc

Add emulate/simulate updated command line options (used for launching the emulation and simulation from the Makefiles/perl scripts)

Change 34769 on 2002/06/18 by efong@efong_r400_win_tor_doc

Updated ... changed overscan to be done

Change 34756 on 2002/06/18 by tho@tho_r400_win

-matrix transform and adjustment complete with debug test
-ycbcr subsampling complete with debug test

Change 34750 on 2002/06/18 by imuskatb@imuskatb_r400_win_cnimuskatb

updated

Change 34747 on 2002/06/18 by imuskatb@imuskatb_r400_win_cnimuskatb

updated

Change 34731 on 2002/06/18 by bryans@bryans_crayola_doc

Update to reflect the additions in TESTINFO object:

getArchitecture()
getConfiguration()
getSimMode()
getSimType()
getTimingMode()
getTestMode()

- removed reference to Architecture object (this is retired for the new chip library)

Change 34730 on 2002/06/18 by imuskatb@imuskatb_r400_win_cnimuskatb

updated

Change 34722 on 2002/06/18 by rfevreau@rfevreau_r400_win

TMDS schedule updated

Change 34690 on 2002/06/18 by efong@efong_r400_win_tor_doc

Updated and put in who is responsible for M2 stuff

Change 34679 on 2002/06/18 by jasif@jasif_r400_win_tor

Updated.

Change 34654 on 2002/06/18 by jacarey@fl_jcarey2

Multiplier in the Micro Engine is 14x14

Change 34629 on 2002/06/17 by jasif@jasif_r400_win_tor

Updated.

Change 34623 on 2002/06/17 by tho@tho_r400_win

update

Change 34614 on 2002/06/17 by jowang@jowang_R400_win

updated horizontal filter (not complete)

Change 34608 on 2002/06/17 by snezana@snezana_crayola_win_cvd

updated

Change 34601 on 2002/06/17 by jasif@jasif_r400_win_tor

Updated.

Change 34591 on 2002/06/17 by rbell@rbell_crayola_win_cvd

updated

Change 34582 on 2002/06/17 by bryans@bryans_crayola_doc

update...

Change 34571 on 2002/06/17 by paulv@MA_PVELLA

Forgot to save changes from previous submission.

Change 34567 on 2002/06/17 by paulv@MA_PVELLA

Updated MHS with latest specifications and hardware changes.

Change 34557 on 2002/06/17 by jasif@jasif_r400_win_tor

Updated.

Change 34552 on 2002/06/17 by jacarey@fl_jcarey2

Remove Slow Client Protocol from the RBBM documentation.

Change 34550 on 2002/06/17 by gabarca@gabarca_crayola_win_cvd

updated table of vga extended registers

Change 34534 on 2002/06/17 by frivas@FL_FRivas

Update to PNL tests

Change 34530 on 2002/06/17 by ashishs@fl_ashishs_r400_win

update

Change 34527 on 2002/06/17 by frivas@FL_FRivas

Update to HOS PNL test.

Change 34515 on 2002/06/17 by frivas@FL_FRivas

Update to HOS PNT using discrete tessellation with lighting and texturing.

Change 34508 on 2002/06/17 by frivas@FL_FRivas

Update to HOS PNT with discrete tessellation and orthographic projection.

Change 34500 on 2002/06/17 by frivas@FL_FRivas

Update to HOS PNT test.

Change 34498 on 2002/06/17 by mkelly@fl_mkelly_r400_win_laptop

Test SC rectangle list vertex order interpretation...

Change 34491 on 2002/06/17 by frivas@FL_FRivas

Update to PNT HOS test that uses only a single PNT in wireframe mode for continuous tessellation that varies between 1.0-14. with reuse 4-16.

Change 34485 on 2002/06/17 by frivas@FL_FRivas

Update to HOS PNT texture and lighting test.

Change 34479 on 2002/06/17 by frivas@FL_FRivas

Update to HOS PNT texture and lighting test

Change 34475 on 2002/06/17 by rramsey@RRAMSEY_P4_r400_win

Add documentation on sc dump points

Change 34457 on 2002/06/17 by tho@tho_r400_win

updated

Change 34454 on 2002/06/17 by frivas@FL_FRivas

Update to HOS PNT test with changing normals.

Change 34452 on 2002/06/17 by jasif@jasif_r400_win_tor

Updated.

Change 34443 on 2002/06/17 by chwang@chwang_doc_r400_win_cvd

Update

Change 34439 on 2002/06/17 by imuskatb@imuskatb_r400_win_cnimuskatb

updated

Change 34438 on 2002/06/17 by imuskatb@imuskatb_r400_win_cnimuskatb

Updated

Change 34437 on 2002/06/17 by jacarey@fl_jcarey2

Remove Constant Flag from the write requesters.

Change 34436 on 2002/06/17 by frivas@FL_FRivas

Update to HOS PNT texture test

Change 34431 on 2002/06/17 by jacarey@fl_jcarey2

1. Set_Constant: Qualified incrementing coherency counter with CONST_WRITE_ENABLE.
2. Updated 2D-to-3D mode determination.

Change 34417 on 2002/06/17 by frivas@FL_FRivas

Update to HOS PNT test that adjusts normals during runtime.

Change 34415 on 2002/06/17 by rbell@rbell_crayola_win_cvd

Changed to testchip.arch and testchip.conf

Change 34414 on 2002/06/17 by chwang@chwang_doc_r400_win_cvd

Update.

Change 34413 on 2002/06/17 by rbell@rbell_crayola_win_cvd

updated

Change 34412 on 2002/06/17 by imuskatb@imuskatb_r400_win_cnimuskatb

Disconnect Test + golden
updated crtc emu doc

Change 34403 on 2002/06/17 by efong@efong_r400_win_tor_doc

Updated

Change 34369 on 2002/06/16 by ashishs@fl_ashishs_r400_win

update

Change 34294 on 2002/06/14 by jasif@jasif_r400_win_tor

Updated.

Change 34290 on 2002/06/14 by csampayo@fl_csampayo_lt_r400

Various document updates

Change 34281 on 2002/06/14 by lkang@lkang_r400_win_tor

updated scaler emulation schedule

Change 34269 on 2002/06/14 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 34264 on 2002/06/14 by nbarbier@nbarbier_r400_win_tor

Updated Auto-detection section.

Change 34242 on 2002/06/14 by kcorrell@kcorrell_r400_docs_marlboro

update

Change 34221 on 2002/06/14 by efong@efong_r400_win_tor_doc

Added in new vertical replication tests

Change 34207 on 2002/06/14 by rvelez@rvelez_r400_win_tor

Updated 6/14

Change 34205 on 2002/06/14 by snezana@snezana_crayola_win_cvd

updated for alu inside framecntl

Change 34183 on 2002/06/14 by efong@efong_r400_win_tor_doc

Just strikethrough on the tap configuration

Change 34146 on 2002/06/14 by mmantor@mmantor_r400_win

moved spec to sc directory and added a test bench document to the directory

Change 34141 on 2002/06/14 by tien@ma_spinach

Updated TP and TC areas

Change 34061 on 2002/06/14 by imuskatb@imuskatb_r400_win_cnimuskatb

Updated

Change 33916 on 2002/06/13 by jasif@jasif_r400_win_tor

Updated.

Change 33910 on 2002/06/13 by mzhu@mzhu_r400_win_tor

add test cases for grphics/overlay blend

Change 33835 on 2002/06/13 by mzhu@mzhu_r400_win_tor

add test plan for second milestone

Change 33820 on 2002/06/13 by hartogs@fl_hartogs

New revision is 0.96
Added diagram to show VGT configurations.
Updated Tessellation Engine fixed function table.

Change 33807 on 2002/06/13 by gabarca@gabarca_crayola_win_cvd

Added in open issues add register bit to select if line compare happens in compared line of the next

Change 33796 on 2002/06/13 by gabarca@gabarca_crayola_win_cvd

updated open issue: readback of VGA DISP interface

Change 33794 on 2002/06/13 by gabarca@gabarca_crayola_win_cvd

Updated open issues and legacy reg list

Change 33757 on 2002/06/13 by lkang@lkang_r400_win_tor

Incorporated emulation, netlist, & chip integration schedules

Change 33619 on 2002/06/12 by imuskatb@imuskatb_r400_win_cnimuskatb

wrap around test
updated display config to support sync_a_start
vga render test V1.0 - test run only

Change 33577 on 2002/06/12 by hartogs@fl_hartogs

New revision number is 0.95
Updated all register definitions to reflect new "driver friendly" address assignments.
This was architecture-wide change made on 5/15/02.
Changed GFX_COPY_STATE register description to reflect actual hardware implementation.
Deleted GFX_PIPE_CNTL register.
Added enumerations to the VGT_EVENT_INITIATOR register.
Added section about restrictions for fields in VGT_GRP_PRIM_TYPE register.
Removed TESS_INPUT_MODE field from VGT_HOS_CNTL register.
Changed VGT_SQ interface (in agreement with Laurent Lefebvre) so that the VGT_SQ_end_of_vtx_vect and VGT_SQ_state signals are "don't care" if the VGT_SQ_continued signal is set.

Change 33532 on 2002/06/12 by frising@ma_frising

Another checkpoint. Start cleaning up usage tables.

Change 33518 on 2002/06/12 by frivas@FL_FRivas

Update - Added a test to HOS PNT section. The new test implements lighting with moving normals.

Change 33490 on 2002/06/12 by frivas@FL_FRivas

Update to HOS PNQ test.

Change 33464 on 2002/06/12 by jasif@jasif_r400_win_tor

Updated.

Change 33362 on 2002/06/11 by frising@ma_frising

Post June 15 1.40 checkpoint.

Change 33349 on 2002/06/11 by ashishs@fl_ashishs_r400_win

update

Change 33339 on 2002/06/11 by frivas@FL_FRivas

Update to PNQ HOS test.

Change 33329 on 2002/06/11 by mkelly@fl_mkelly_r400_win_laptop

update, sc point tests....

Change 33299 on 2002/06/11 by jayw@MA_JAYW

more FB and cache line formats

Change 33282 on 2002/06/11 by frivas@FL_FRivas

Update to HOS test for single PNT with cubic position and quadratic normal using continuous tessellation.

Change 33275 on 2002/06/11 by tho@tho_r400_win

updated, matrix transform and adjustment emulation block completed

Change 33219 on 2002/06/11 by nbarbier@nbarbier_r400_win_tor

Updated description of TMDS reduction block.

Change 33215 on 2002/06/11 by frivas@FL_FRivas

Update to HOS test of a single PNT with linear interpolation and continuous tessellation.

Change 33201 on 2002/06/11 by frivas@FL_FRivas

Update to status of HOS test of PNT's with lighting and texturing.

Change 33199 on 2002/06/11 by frivas@FL_FRivas

Update to HOS tests for continuous tessellation of PNT's

Change 33195 on 2002/06/11 by csampayo@fl_csampayo_r400

Edited Windows Registry, Test Format and Test Samples sections

Change 33177 on 2002/06/11 by jowang@jowang_R400_win

modified spreadsheet for 10 tap H
updated lbscfp to output fixed point coefs to file

Change 33172 on 2002/06/11 by csampayo@fl_csampayo_r400

Update status for the following test:
r400vgt_hos_PNT_cp_qn_disc_14_04_lit_tex_proj_01

Change 33148 on 2002/06/11 by frivas@FL_FRivas

Update to HOS tests of PNT and PNL

Change 33147 on 2002/06/11 by frivas@FL_FRivas

Progress on HOS tests documented.

Change 33098 on 2002/06/10 by bryans@bryans_crayola_doc

Update based on today's discussion

Change 33070 on 2002/06/10 by tho@tho_r400_win

icon cursor testlist updated

Change 33054 on 2002/06/10 by mkelly@fl_mkelly_r400_win_laptop

Update, sc tests...

Change 33041 on 2002/06/10 by jayw@MA_JAYW

initial checkin

Change 33040 on 2002/06/10 by jayw@MA_JAYW

several 'admin' type signals
clean and busy
resets
clocks
removed shadow from queue encoding.

Change 33029 on 2002/06/10 by jacarey@fl_jcarey2

Check In All Files.

Change 33023 on 2002/06/10 by vromaker@MA_VIC_P4

updated sq top block diagram

Change 33021 on 2002/06/10 by ashishs@fl_ashishs_r400_win

update

Change 33002 on 2002/06/10 by jacarey@fl_jcarey2

Added RT_GUI_ACTIVE to rbbm_status

Change 32990 on 2002/06/10 by nluu@nluu_r400_doclib_cnnb

- update

Change 32988 on 2002/06/10 by jacarey@fl_jcarey2

Remove hirq_pending from gui_active determination.

Change 32987 on 2002/06/10 by vliu@vliu_r400_cnvliu100_win_cvd

Update

Change 32981 on 2002/06/10 by jacarey@fl_jcarey2

Fix typo in sub-blk_prefetch packet.
DWORD count should be 7:0 because the largest size is 128 DWORDs.

Change 32973 on 2002/06/10 by jacarey@fl_jcarey2

Add descriptions of packets for after June15th Milestone.

Change 32963 on 2002/06/10 by csampayo@fl_csampayo_r400

Updated status for the following tests:
r400vgt_line_list_01
r400vgt_quad_strip_01
r400vgt_quad_list_01
r400vgt_rectangle_list_01
r400vgt_polygon_01
r400vgt_line_loop_01
r400vgt_point_list_01
r400vgt_line_strip_01
r400vgt_triangle_fan_01
r400vgt_triangle_list_01
r400vgt_triangle_strip_01
r400vgt_triangle_wflags_01

Change 32960 on 2002/06/10 by tho@tho_r400_win

updated

Change 32947 on 2002/06/10 by chwang@chwang_doc_r400_win_cvd

Update. Added self to Icon and Cursor for test API.

Change 32939 on 2002/06/10 by jasif@jasif_r400_win_tor

Updated.

Change 32935 on 2002/06/10 by jasif@jasif_r400_win_tor

Updated.

Change 32921 on 2002/06/10 by rvelez@rvelez_r400_win_tor

Updated 6/10

Change 32907 on 2002/06/10 by jacarey@fl_jcarey2

Clarification on CMDFIFO Entries in Wait_Until

Change 32903 on 2002/06/10 by rbell@rbell_crayola_win_cvd

updated

Change 32899 on 2002/06/10 by rbell@rbell_crayola_win_cvd

updated

Change 32897 on 2002/06/10 by imuskatb@imuskatb_r400_win_cnimuskatb

Updated

Change 32872 on 2002/06/09 by jimmylau@jimmylau_r400_win_tor

Major update of DCCG specs after specs review.

Change 32809 on 2002/06/07 by efong@efong_r400_win_tor_doc

updated

Change 32807 on 2002/06/07 by efong@efong_r400_win_tor_doc

updated

Change 32805 on 2002/06/07 by rvelez@rvelez_r400_win_tor

Updated 6/7

Change 32785 on 2002/06/07 by jhoule@MA_JHOULE

Added separable variation, showing it still works.
Weights stored now mean something else (a little bit).

Change 32749 on 2002/06/07 by dclifton@dclifton_r400

Added description of VTE and VE

Change 32729 on 2002/06/07 by jasif@jasif_r400_win_tor

Updated.

Change 32718 on 2002/06/07 by jacarey@fl_jcarey2

Added missing signal: CG_CP_pm_enb to external interfaces.

Change 32683 on 2002/06/07 by efong@efong_r400_win_tor_doc

Updated

Change 32679 on 2002/06/07 by jayw@MA_JAYW

new depth blocks

Change 32662 on 2002/06/07 by jacarey@fl_jcarey2

Add note on 2D/3D transition detection to PFP section.

Change 32653 on 2002/06/07 by jacarey@fl_jcarey2

Document method of determining 2D/3D transitions....

Change 32647 on 2002/06/07 by bryans@bryans_crayola_doc

Update with new top level targets:

gen_block_list
detailed_summary

Add: block_list.txt description to appendix

Change 32643 on 2002/06/07 by dclifton@dclifton_r400

Update with increased z precision, latest I/O and register spec.

Change 32641 on 2002/06/07 by jacarey@fl_jcarey2

Add 2D/3D Mode switch assumptions to pseudocode.

Change 32621 on 2002/06/07 by jasif@jasif_r400_win_tor

Updated.

Change 32619 on 2002/06/07 by jacarey@fl_jcarey2

Update CMDFIFO_AVAIL for RBBM_STATUS

Change 32609 on 2002/06/07 by jasif@jasif_r400_win_tor

Updated.

Change 32600 on 2002/06/07 by jacarey@fl_jcarey2

Update Soft Reset Register Text

Change 32582 on 2002/06/07 by jhoule@MA_JHOULE

Arbitrary filter weight description.

Change 32574 on 2002/06/07 by vgoel@fl_vgoel2

added description for test r400vgt_hos_PNT_disc_cp_qn_14_4_light_texture_stress.cpp and changed status of (% done) for couple of tests.

Change 32569 on 2002/06/07 by lseiler@lseiler_r400_win_marlboro

Changed SX_RB_mask_type to SX_RB_quad_type, removed RB_SX_index_next, added text and SX_RB signals for new memory export method.

Change 32562 on 2002/06/07 by gregs@gregs_r400_win_marlboro

removed 3 of the CG_CP RT signals.

Change 32473 on 2002/06/06 by georgev@ma_georgev

First cut.

Change 32451 on 2002/06/06 by bryans@bryans_crayola_doc

Update status - add DCCIF enhancement request

Change 32444 on 2002/06/06 by askende@andi_r400_docs

updated the RB_SX_index interface to include "RB_SX_index_op"

Change 32443 on 2002/06/06 by rbell@rbell_crayola_win_cvd

Added keyer tests

Change 32438 on 2002/06/06 by mzhu@mzhu_r400_win_tor

Update description for ICON and CURSOR width and height registers.

Change 32433 on 2002/06/06 by vgoel@fl_vgoel2

removed a repeated test and changed PNL test to require linear normal instead of quadratic normal

Change 32428 on 2002/06/06 by vgoel@fl_vgoel2

added status of HOS test (% done)

Change 32374 on 2002/06/06 by jacarey@fl_jcarey2

Fix Typo in Section Name

Change 32373 on 2002/06/06 by gregs@gregs_r400_linux_marlboro

vcs compiles.

Change 32369 on 2002/06/06 by jacarey@fl_jcarey2

1. Add Section for 2D Indirect Buffer Contents to 2D Appendix
2. Updated CP Initialization Sequence.

Change 32353 on 2002/06/06 by jacarey@fl_jcarey2

Fix Typo in the Trans_Bitblt packet description

Change 32350 on 2002/06/06 by markf@markf_r400_win_marlboro

Updated Status

Change 32334 on 2002/06/06 by semara@semara_r400_win_tor

update for the text section

Change 32300 on 2002/06/06 by rbell@rbell_crayola_win_cvd

update after review

Change 32289 on 2002/06/06 by vgoel@fl_vgoel2

changed description of tests for continuous tessellation of PNT, PNQ and PNL.

Change 32283 on 2002/06/06 by lseiler@lseiler_r400_win_marlboro

Updated pdf to match current MemCtl.doc

Change 32279 on 2002/06/06 by dclifton@dclifton_r400

Connected up some rcpeng signals

Change 32274 on 2002/06/06 by lseiler@lseiler_r400_win_marlboro

Moved and renamed to design/chip/memory/R400_Memory*

Change 32272 on 2002/06/06 by lseiler@lseiler_r400_win_marlboro

Moved to design/chip/memory

Change 32271 on 2002/06/06 by lseiler@lseiler_r400_win_marlboro

Moved here from design/blocks/mc

Change 32265 on 2002/06/06 by vgoel@fl_vgoel2

Changed discrete tessellation tests.

Change 32166 on 2002/06/05 by csampayo@fl_csampayo_r400

Updated status for the following VGT tests:
r400vgt_index_offset_01
r400vgt_index_offset_02
r400vgt_index_offset_03

Change 32156 on 2002/06/05 by rvelez@rvelez_r400_win_tor

Updated 6/5

Change 32143 on 2002/06/05 by csampayo@fl_csampayo_r400

Updated status for the following VGT tests:
r400vgt_index_dealloc_points_01
r400vgt_index_dealloc_line_list_01
r400vgt_index_dealloc_triangle_list_01

Change 32140 on 2002/06/05 by gregs@gregs_r400_linux_marlboro

display section updated for r400.

Change 32122 on 2002/06/05 by jacarey@fl_jcarey2

Add more qualifications to the generation of the destination load signals.

Change 32114 on 2002/06/05 by tho@tho_r400_win

updated

Change 32097 on 2002/06/05 by jacarey@fl_jcarey2

Remove queued_path_busy from rbbm_status register.

Change 32080 on 2002/06/05 by jacarey@fl_jcarey2

Removed Queued_Path_Busy from the RBBM_STATUS Register

Change 32073 on 2002/06/05 by gregs@gregs_r400_win_marlboro

update

Change 32061 on 2002/06/05 by nbarbier@nbarbier_r400_win_tor

Updated TMDSA section.

Change 32055 on 2002/06/05 by rbell@rbell_crayola_win_cvd

Added overlay/VP window swap tests

Change 32040 on 2002/06/05 by vgoel@fl_vgoel2

Created descriptions for HOS tests : PNT (discrete, continuous), PNQ (continuous), PNL (continuous)

Change 31999 on 2002/06/05 by jacarey@fl_jcarey2

Added CP_NRT_BUSY to CP_STAT register

Change 31979 on 2002/06/05 by jacarey@fl_jcarey2

RBBM Spec Updates

Change 31964 on 2002/06/05 by mmantor@mmantor_r400_win

added initial sc_packer code to the sc.v and a test bench for it

Change 31919 on 2002/06/05 by jacarey@fl_jcarey2

Miscellaneous Updates to Diagram to Match the Design.

Change 31878 on 2002/06/04 by lseiler@lseiler_r400_win_marlboro

Rev 0.8 Memory Format document with revisions removed

Change 31877 on 2002/06/04 by lseiler@lseiler_r400_win_marlboro

Memory Format spec: extensive revision, also moved from the MC block to a chip-level location

Change 31859 on 2002/06/04 by rbell@rbell_crayola_win_cvd

updated

Change 31858 on 2002/06/04 by gregs@gregs_r400_linux_marlboro

added parallel ROM muxes ...

Change 31798 on 2002/06/04 by jimmylau@jimmylau_r400_win_tor

fix the problem that DCCG top level diagram cannot be accessed from the DCCG specs.
add sections 3.1, 3.2 and 3.3.

Change 31794 on 2002/06/04 by jacarey@fl_jcarey2

1. Added TOC
2. Added clarification of location for vertex and Pixel ALU constants for 2D processing.

Change 31779 on 2002/06/04 by rbell@rbell_crayola_win_cvd

updated

Change 31748 on 2002/06/04 by frivas@FL_FRivas

Entered 4 tests for HOS in Sheet 1, section 1.5.1.1.9.6

Change 31740 on 2002/06/04 by bryans@bryans_crayola_doc

Renamed LB_feature_list.doc to LB_emu_schedule.doc

Change 31737 on 2002/06/04 by jacarey@fl_jcarey2

1. Remove Unneeded Booleans
2. Remove Clamping from ALU
3. Temporary Depth Increase for Microcode RAM

Change 31730 on 2002/06/04 by jacarey@fl_jcarey2

Clarifications to which constants to write.

Change 31713 on 2002/06/04 by vgoel@fl_vgoel2

no change

Change 31711 on 2002/06/04 by markf@markf_r400_win_marlboro

nop

Change 31707 on 2002/06/04 by markf@markf_r400_win_marlboro

Initial rev

Change 31701 on 2002/06/04 by csampayo@fl_csampayo_r400

Updated schedule

Change 31695 on 2002/06/04 by jimmylau@jimmylau_r400_win_tor

Initial Revision of DCCG specs with diagrams for R400.

Change 31680 on 2002/06/04 by smoss@smoss_crayola_win

new tests

Change 31674 on 2002/06/04 by mkelly@fl_mkelly_r400_win

Update

Change 31662 on 2002/06/04 by efong@efong_r400_win_tor_doc

added in another kludge

Change 31607 on 2002/06/03 by frivas@FL_FRivas

Number of tests written updated.

Change 31592 on 2002/06/03 by jacarey@fl_jcarey2

Checkpoint All Files

Change 31591 on 2002/06/03 by jacarey@fl_jcarey2

Fix Typos for Brush Decode Text

Change 31580 on 2002/06/03 by csampayo@fl_csampayo_r400

Updated hos test description

Change 31567 on 2002/06/03 by frising@ma_frising

Added post June15 staging version of texture insts/consts.

Change 31560 on 2002/06/03 by gregs@gregs_r400_linux_marlboro

vip + most of dvo interfaces.

Change 31543 on 2002/06/03 by omesh@ma_omesh

Updated top level spreadsheet with latest information about RB Color blending test information. I was asked to defer the fog related tests to later and Yung had agreed to do the "No Blend" related tests as they are related to Shader Pipe.
Also added blending test names and new section on stress testing to my RBC document.

Change 31529 on 2002/06/03 by bryans@bryans_crayola_doc

Initial regression distribution schedule for TO

Change 31506 on 2002/06/03 by gregs@gregs_r400_win_marlboro

DVODATA is 24 bits wide.

Change 31476 on 2002/06/03 by dclifton@dclifton_r400

Update to include changes for Z precision, major change in I, J, and W gradient calculation. Update to I/O.

Change 31475 on 2002/06/03 by jacarey@fl_jcarey2

1. Removed CP_DMA_Busy as a boolean
2. Result[31:0] is write data for scratch memory.

Change 31473 on 2002/06/03 by jacarey@fl_jcarey2

Update DMA Data Path

Change 31465 on 2002/06/03 by jayw@MA_JAYW

added sx export buffers register
updated several lesser items

Change 31454 on 2002/06/03 by jasif@jasif_r400_win_tor

Updated.

Change 31453 on 2002/06/03 by jasif@jasif_r400_win_tor

Updated.

Change 31433 on 2002/06/03 by llefebvr@llefebvre_laptop_r400

Backup and minor updates.

Change 31432 on 2002/06/03 by tho@tho_r400_win

updated

Change 31422 on 2002/06/03 by efong@efong_r400_win_tor_doc

Updated

Change 31415 on 2002/06/03 by csampayo@fl_csampayo_r400

Updated status for the following test:
r400vgt_hos_cubic_pos_pnt_discrete_01

Change 31409 on 2002/06/03 by chwang@chwang_doc_r400_win_cvd

Update.

Change 31401 on 2002/06/03 by rbell@rbell_crayola_win_cvd

updated

Change 31395 on 2002/06/03 by jacarey@fl_jcarey2

Checkpoint Planning Spreadsheets

Change 31384 on 2002/06/03 by gregs@gregs_r400_win_marlboro

Updated rom_clk register

Change 31275 on 2002/05/31 by rvelez@rvelez_r400_win_tor

Updated 5/31

Change 31273 on 2002/05/31 by mkelly@fl_mkelly_r400_win_laptop

Basic diamond exit tests....

Change 31262 on 2002/05/31 by gabarca@gabarca_crayola_win_cvd

Added GENS1 and GENFC_WT to list od mono/color registers

Change 31248 on 2002/05/31 by jacarey@fl_jcarey2

Misc. Updates to RBIU Diagram

Change 31228 on 2002/05/31 by tho@tho_r400_win

updated testlist

Change 31210 on 2002/05/31 by csampayo@fl_csampayo_r400

Adding Francisco (HOS)

Change 31194 on 2002/05/31 by gabarca@gabarca_crayola_win_cvd

Write combining done by DCCIF
Skid buffer between BIF and VGAHDP

Change 31169 on 2002/05/31 by jayw@MA_JAYW

register index and some formatting

Change 31160 on 2002/05/31 by semara@semara_r400_win_tor

re-submit

Change 31144 on 2002/05/31 by gabarca@gabarca_crayola_win_cvd

Moved write combiming to DCCIF, added skid buffer in BIF_VGA doc

Change 31097 on 2002/05/31 by vliu@vliu_r400_cnvliu100_win_cvd

Update

Change 31096 on 2002/05/31 by gabarca@gabarca_crayola_win_cvd

Updated VGAHDP section

Change 31082 on 2002/05/31 by rbagley@rbagley_ltxp

Adding Jocelyn's recent revision.

Change 31076 on 2002/05/31 by jacarey@fl_jcarey2

Added Source Select to Read Input

Change 31044 on 2002/05/31 by efong@efong_r400_win_tor_doc

Updated

Change 31043 on 2002/05/31 by bryans@bryans_crayola_doc

Update...

Change 31034 on 2002/05/31 by jacarey@fl_jcarey2

Make Source Select a Counter which is controlled by Micro Engine.

Change 31026 on 2002/05/31 by jasif@jasif_r400_win_tor

Updated.

Change 31023 on 2002/05/31 by efong@efong_r400_win_tor_doc

put in another problem

Change 30961 on 2002/05/30 by vliu@vliu_r400_cnvliu100_win_cvd

Update

Change 30959 on 2002/05/30 by jacarey@fl_jcarey2

Add hook to generate real-time microcode...

Change 30955 on 2002/05/30 by jacarey@fl_jcarey2

Fix name of context_done event used for state management.

Change 30934 on 2002/05/30 by jacarey@fl_jcarey2

Fix typo in instr_prefetch packet

Change 30909 on 2002/05/30 by jacarey@fl_jcarey2

Update for error checking.

Change 30893 on 2002/05/30 by jacarey@fl_jcarey2

Fix Typos

Change 30886 on 2002/05/30 by jacarey@fl_jcarey2

Updates for Protected Mode Error Checking...

Change 30864 on 2002/05/30 by jacarey@fl_jcarey2

Update for Conditional Continue Micro Instruction

Change 30854 on 2002/05/30 by rbell@rbell_crayola_win_cvd

updated

Change 30853 on 2002/05/30 by ashishs@fl_ashishs_r400_win

update

Change 30814 on 2002/05/30 by bryans@bryans_crayola_doc

Update for week of May 27

Change 30804 on 2002/05/30 by gregs@gregs_r400_win_marlboro

updated regs.

Change 30750 on 2002/05/29 by vliu@vliu_r400_cnvliu100_win_cvd

Update

Change 30747 on 2002/05/29 by vliu@vliu_r400_cnvliu100_win_cvd

Update

Change 30734 on 2002/05/29 by vliu@vliu_r400_cnvliu100_win_cvd

Update

Change 30732 on 2002/05/29 by vliu@vliu_r400_cnvliu100_win_cvd

Update

Change 30722 on 2002/05/29 by tho@tho_r400_win

test list for icon and cursor

Change 30715 on 2002/05/29 by jowang@jowang_R400_win

Includes Vertical Filter control

Change 30713 on 2002/05/29 by rbell@rbell_crayola_win_cvd

updated

Change 30696 on 2002/05/29 by gabarca@gabarca_crayola_win_cvd

Put VGAHDP tests

Change 30694 on 2002/05/29 by jacarey@fl_jcarey2

Added GMC to opcodes for new 2D packets.

Change 30673 on 2002/05/29 by jayw@MA_JAYW

for larry's review

Change 30626 on 2002/05/29 by jhoule@MA_JHOULE

0.9.12: Update all the external interfaces.

Change 30615 on 2002/05/29 by rbell@rbell_crayola_win_cvd

updated

Change 30609 on 2002/05/29 by rbell@rbell_crayola_win_cvd

Added Milestone 2 tests

Change 30601 on 2002/05/29 by jayw@MA_JAYW

ROP3 and Chroma Keying

Change 30595 on 2002/05/29 by gregs@gregs_r400_win_marlboro

pm software review.

Change 30588 on 2002/05/29 by lseiler@lseiler_r400_win_marlboro

Memory Format spec: final submission before changing file name and location.

Change 30546 on 2002/05/29 by jacarey@fl_jcarey2

Remove reference of "Go" signals from the RBBM Spec.
This was a typo.

Change 30502 on 2002/05/29 by jacarey@fl_jcarey2

Change COND_MEM_WRITE TO COND_WRITE

Change 30466 on 2002/05/28 by gabarca@gabarca_crayola_win_cvd

Updated write modes

Change 30448 on 2002/05/28 by jasif@jasif_r400_win_tor

Updated.

Change 30415 on 2002/05/28 by jacarey@fl_jcarey2

Removed CP_CG_2D_Mode signal per power management meeting...

Change 30406 on 2002/05/28 by khabbari@khabbari_r400_win

auto coef cal added

Change 30401 on 2002/05/28 by efong@efong_r400_win_tor_doc

Added in Kaleidoscope Kludge document

Change 30348 on 2002/05/28 by rbell@rbell_crayola_win_cvd

updated

Change 30342 on 2002/05/28 by jacarey@fl_jcarey2

1. Added 8 more bios scratch registers
2. DMA engine waits for MH to accept last write transaction before raising the interrupt.

Change 30277 on 2002/05/28 by ashishs@fl_ashishs_r400_win

update

Change 30191 on 2002/05/27 by gabarca@gabarca_crayola_win_cvd

Added more description to VGAHDP features

Change 30178 on 2002/05/27 by imuskatb@imuskatb_r400_win_cnimuskatb

Updated

Change 30157 on 2002/05/27 by rvelez@rvelez_r400_win_tor

Updated 5/27

Change 30152 on 2002/05/27 by gabarca@gabarca_crayola_win_cvd

Updated Worksheet

Change 30151 on 2002/05/27 by gabarca@gabarca_crayola_win_cvd

Updated worsheet

Change 30148 on 2002/05/27 by gabarca@gabarca_crayola_win_cvd

Updated VGA modes spreadsheet

Change 30143 on 2002/05/27 by efong@efong_r400_win_tor_doc

Updated

Change 30140 on 2002/05/27 by gabarca@gabarca_crayola_win_cvd

updated excel spreadsheet whit modes

Change 30139 on 2002/05/27 by chwang@chwang_doc_r400_win_cvd

Update (nothing listed).

Change 30089 on 2002/05/25 by jimmylau@jimmylau_r400_win_tor

Add test cases to verify composite H sync and also feature to disable H sync cutoff in section 5.1.1.
Add section 5.4 to test dual CRTCs in progressive scanning mode.

Change 30088 on 2002/05/25 by jimmylau@jimmylau_r400_win_tor

Explain the algorithm to generate composite H sync, under section 6.2

Change 30067 on 2002/05/24 by mmang@fl_mmang_r400_win

Updated spec for design review.

Change 30055 on 2002/05/24 by gabarca@gabarca_crayola_win_cvd

Cleaned up, added tables

Change 30020 on 2002/05/24 by jacarey@fl_jcarey2

Check everything in...

Change 30019 on 2002/05/24 by nbarbier@nbarbier_r400_win_tor

Updated TMDS, DVO & HPD sections.

Change 29999 on 2002/05/24 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 29936 on 2002/05/24 by rvelez@rvelez_r400_win_tor

Updated 5/24

Change 29926 on 2002/05/24 by jacarey@fl_jcarey2

Update packet restrictions for Real-Time Processing.

Change 29916 on 2002/05/24 by jacarey@fl_jcarey2

Updates to clarify what to do for real-time stream processing.

Change 29909 on 2002/05/24 by gregs@gregs_r400_win_marlboro

added to PM programming guide (still under construction).

Change 29907 on 2002/05/24 by jacarey@fl_jcarey2

Add "invert" to real-time signal conditioning selects.

Change 29884 on 2002/05/24 by rherrick@ma_rherrick_crayola

Once again, I think I submitted before WORD had actually written the changes to the disk... Hopefully this time it makes it in correctly...

Change 29876 on 2002/05/24 by rherrick@ma_rherrick_crayola

Update spec to reflect RB_ACCESS_TYPE modes implemented in the parser..

Change 29794 on 2002/05/23 by fhsien@fhsien_r400_win_marlboro

UPDATE NAME

Change 29784 on 2002/05/23 by gregs@gregs_r400_win_marlboro

programmer's guide - initial.

Change 29733 on 2002/05/23 by tho@tho_r400_win

updated

Change 29703 on 2002/05/23 by jacarey@fl_jcarey2

Update Compare Function of CP_RT*_Command Register

Change 29691 on 2002/05/23 by jacarey@fl_jcarey2

Update Status and Planning Documents

Change 29643 on 2002/05/23 by jacarey@fl_jcarey2

Document Updates

Change 29613 on 2002/05/23 by smoss@smoss_crayola_win

su test

Change 29593 on 2002/05/22 by jimmylau@jimmylau_r400_win_tor

Fill in section for CG interface.

Change 29577 on 2002/05/22 by imuskatb@imuskatb_r400_win_cnimuskatb

New Dispout reg test
Updated CRTC reg test
Updated Dispout xls doc

Change 29563 on 2002/05/22 by rvelez@rvelez_r400_win_tor

Updated 5/22

Change 29558 on 2002/05/22 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 29553 on 2002/05/22 by jacarey@fl_jcarey2

Checkpoint Validation Plan and To Liest

Change 29506 on 2002/05/22 by jacarey@fl_jcarey2

Clarified that Sub-block Offset is DWORD offset for ME_INTI.

Change 29488 on 2002/05/22 by snezana@snezana_crayola_win_cvd

small update, no real change

Change 29485 on 2002/05/22 by nbarbier@nbarbier_r400_win_tor

Updated Macro Interface & IO Interface.

Change 29484 on 2002/05/22 by gregs@gregs_r400_win_marlboro

Core Sclk clock PLL connection table + minor updates.

Change 29470 on 2002/05/22 by jacarey@fl_jcarey2

Clarified Swap Controls

Change 29449 on 2002/05/22 by jacarey@fl_jcarey2

Added RBBM_STATUS2 Register
Added CP_RBBM_rt_enable to RBBM_STATUS register

Change 29438 on 2002/05/22 by jacarey@fl_jcarey2

Clarify that DST_TYPE=7 not used in R400...since R128

Change 29437 on 2002/05/22 by jacarey@fl_jcarey2

Corrections for 2D GMC Processing.

Change 29358 on 2002/05/21 by lseiler@lseiler_r400_win_marlboro

RB Spec: Updated SX section, other updates in process

Change 29343 on 2002/05/21 by jacarey@fl_jcarey2

Update CP Specs to Reflect 4KByte Alignment for Surfaces.

Change 29335 on 2002/05/21 by imuskatb@imuskatb_r400_win_cnimuskatb

updated crc test
first crtc register test

Change 29322 on 2002/05/21 by rvelez@rvelez_r400_win_tor

Added +l to disallow multiple file checkouts

Change 29320 on 2002/05/21 by efong@efong_r400_win_tor_doc

changed it to +l to allow for locking by just one person

Change 29310 on 2002/05/21 by askende@andi_r400_docs

fixed typo

Change 29305 on 2002/05/21 by jacarey@fl_jcarey2

Clarify compare function for Cond_Mem_Write and Wait_Reg_Mem

Change 29286 on 2002/05/21 by jacarey@fl_jcarey2

Clarification of signals used by the Wait_Until and RTS_WAIT_Until registers.

Change 29275 on 2002/05/21 by rvelez@rvelez_r400_win_tor

Updated 5/17

Change 29272 on 2002/05/21 by tho@tho_r400_win

updated

Change 29262 on 2002/05/21 by rvelez@rvelez_r400_win_tor

Updated Top-level DC interface to match source code 5/21

Change 29256 on 2002/05/21 by chwang@chwang_doc_r400_win_cvd

Weekly update.

Change 29237 on 2002/05/21 by efong@efong_r400_win_tor_doc

Updated to be the same as the test plan

Change 29234 on 2002/05/21 by jasif@jasif_r400_win_tor

Updated status.

Change 29231 on 2002/05/21 by jacarey@fl_jcarey2

Updated compare function bit width for Cond_Mem_Write and Wait_Reg_Mem.
Text Updates to MPEG_Index packet

Change 29227 on 2002/05/21 by efong@efong_r400_win_tor_doc

changed from x130 to x132

Change 29223 on 2002/05/21 by efong@efong_r400_win_tor_doc

Updated

Change 29199 on 2002/05/21 by jacarey@fl_jcarey2

Updates to register descriptions and fix typos.

Change 29108 on 2002/05/20 by pmitchel@pmitchel_entire_depot_win

add to depot

Change 29100 on 2002/05/20 by pmitchel@pmitchel_entire_depot_win

adding file to depot

Change 29071 on 2002/05/20 by gregs@gregs_r400_win_marlboro

added few gnd/pwr pins for TMDS and DACs.

Change 29061 on 2002/05/20 by csampayo@fl_csampayo_r400

Udated status for the following test:
r400vgt_hos_simple_linear_PNT_discrete_01

Change 29012 on 2002/05/20 by kcorrell@kcorrell_r400_docs_marlboro

updated tc interface and display interface

Change 29003 on 2002/05/20 by gregs@gregs_r400_win_marlboro

update

Change 28998 on 2002/05/20 by jacarey@fl_jcarey2

Fix Typos in the MPEG_INDEX packet.

Change 28980 on 2002/05/19 by fhsien@fhsien_r400_win_marlboro

Update

Change 28859 on 2002/05/17 by mzhu@mzhu_r400_win_tor

Add LUT bypassing registers. Update LUT register descriptions

Change 28801 on 2002/05/17 by smoss@smoss_crayola_win

SU tests with golds

Change 28787 on 2002/05/17 by jacarey@fl_jcarey2

Updates to:
1. Wait_Reg_Mem
2. Cond_Mem_Write

Change 28768 on 2002/05/17 by kmahler@kmahler_r400_doc_lib

Revised Paramtterized Test Case chapter to inlcude changes for specifying the Test Case Number and other enhancements.

Added section "Specify Input Arguments to the Test Program".

Change 28720 on 2002/05/17 by jacarey@fl_jcarey2

No Clamping required by the CP for UR & LL compound indices.

Change 28718 on 2002/05/17 by vliu@vliu_r400_cnvliu100_win_cvd

Update

Change 28711 on 2002/05/17 by jacarey@fl_jcarey2

Remove reference to non-queued path for BIF transaction.

Change 28679 on 2002/05/17 by fhsien@fhsien_r400_win_marlboro

Update RBD

Change 28662 on 2002/05/17 by vliu@vliu_r400_cnvliu100_win_cvd

Reformat

Change 28651 on 2002/05/17 by vliu@vliu_r400_cnvliu100_win_cvd

Update

Change 28572 on 2002/05/16 by csampayo@fl_csampayo_r400

Updated status for the following tests:
r400vgt_dma_swap_indx16_01
r400vgt_pass_thru_all_prims_01

Change 28562 on 2002/05/16 by vliu@vliu_r400_cnvliu100_win_cvd

Update

Change 28532 on 2002/05/16 by vliu@vliu_r400_cnvliu100_win_cvd

Update

Change 28526 on 2002/05/16 by vliu@vliu_r400_cnvliu100_win_cvd

Update

Change 28512 on 2002/05/16 by jacarey@fl_jcarey2

Max Context resets to 7

Change 28507 on 2002/05/16 by jacarey@fl_jcarey2

Misc Updates

Change 28493 on 2002/05/16 by rbell@rbell_crayola_win_cvd

updated

Change 28483 on 2002/05/16 by jacarey@fl_jcarey2

Add Max Context to address

Change 28455 on 2002/05/16 by vliu@vliu_r400_cnvliu100_win_cvd

Update

Change 28437 on 2002/05/16 by vliu@vliu_r400_cnvliu100_win_cvd

Update

Change 28435 on 2002/05/16 by gregs@gregs_r400_win_marlboro

added interrupt registers.

Change 28429 on 2002/05/16 by vliu@vliu_r400_cnvliu100_win_cvd

Update

Change 28423 on 2002/05/16 by jacarey@fl_jacarey

Update All

Change 28418 on 2002/05/16 by jacarey@fl_jacarey

Update source clipping equation

Change 28412 on 2002/05/16 by jimmylau@jimmylau_r400_win_tor

Fine adjust the parameter values in normal CRTC tests in section 5.1.1

Change 28404 on 2002/05/16 by ashishs@fl_ashishs_r400_win

update

Change 28392 on 2002/05/16 by omesh@ma_omesh

Added some more comments for the test group levels, to specify the goal of each group of tests.

Change 28385 on 2002/05/16 by rbell@rbell_crayola_win_cvd

Updated

Change 28367 on 2002/05/16 by jasif@jasif_r400_win_tor

Updated after VGA testplan review.

Change 28362 on 2002/05/16 by omesh@ma_omesh

Checking in top-level details of RBC related tests. Document (*.doc) links are not yet in place, as document to be linked (RBC.doc) is not yet "section labelled".

Change 28348 on 2002/05/16 by bryans@bryans_crayola_doc

Weekly DV goals for Video IP (Summary)

Change 28327 on 2002/05/16 by jacarey@fl_jacarey

Checkpoint Everything

Change 28325 on 2002/05/16 by jacarey@fl_jacarey

Microcode Update
Update 2D Appendix for source clipping
Update PM4 spec for Reg-to-Mem Code

Change 28275 on 2002/05/15 by fhsien@fhsien_r400_win_marlboro

Update RBD part of list

Change 28261 on 2002/05/15 by bryans@bryans_crayola_doc

Change emulation to M1/M2/M3 lineup

Change 28256 on 2002/05/15 by omesh@ma_omesh

Made some corrections to misleading wording and added some more information to document.

Change 28232 on 2002/05/15 by jacarey@fl_jacarey

Update Number Formats for the 2D Packets.

Change 28224 on 2002/05/15 by fhsien@fhsien_r400_win_marlboro

Update RBD testplan

Change 28223 on 2002/05/15 by jacarey@fl_jacarey

Fix typo in Width format -- changed to 13:0

Change 28196 on 2002/05/15 by jacarey@fl_jacarey

Update Top-Level Diagrams

Change 28149 on 2002/05/15 by gregs@gregs_r400_win_marlboro

general update

Change 28148 on 2002/05/15 by rbell@rbell_crayola_win_cvd

Updated test plan

Change 28138 on 2002/05/15 by vliu@vliu_r400_cnvliu100_win_cvd

Update

Change 28130 on 2002/05/15 by gabarca@gabarca_crayola_win_cvd

Finished for Milestone 1, in bold

Change 28104 on 2002/05/15 by kmahler@kmahler_r400_doc_lib

Added more detailed explanation of using RENDER_STATE's low-level interface per Carlos' note.

Change 28098 on 2002/05/15 by jasif@jasif_r400_win_tor

Added testlist.

Change 28095 on 2002/05/15 by georgev@ma_georgev

Examples added.

Change 28092 on 2002/05/15 by csampayo@fl_csampayo_r400

Updated status for the following test:
r400vgt_dma_swap_indx32_01

Change 28087 on 2002/05/15 by omesh@ma_omesh

Added some more stuff.... Document still incomplete with lots of areas of construction. Will add test total counts today and update spreadsheet....

Change 28055 on 2002/05/15 by imuskatb@imuskatb_r400_win_cnimuskatb

add new crtc test under debug
updated crtc doc

Change 28047 on 2002/05/15 by jacarey@fl_jacarey

1. CP Spec : Update Real-Time Register Names
2. PM4 Spec : Updates to ME_Init and Event_Timestamp_Write

Change 28046 on 2002/05/15 by vliu@vliu_r400_cnvliu100_win_cvd

Update

Change 28011 on 2002/05/14 by kmahler@kmahler_r400_doc_lib

Added documentation for the following:

1) Revised Parameterized Test Cases using input arguments and the "Set_Default_Values()" function.

2) VFD section from Michael Kelly.

Change 28006 on 2002/05/14 by imuskatb@imuskatb_r400_win_cnimuskatb

updated

Change 28004 on 2002/05/14 by gabarca@gabarca_crayola_win_cvd

tried to finish

Change 28003 on 2002/05/14 by mzhu@mzhu_r400_win_tor

move dcp test plan to design/blocks directory
add test plan for first milestone

Change 27980 on 2002/05/14 by jacarey@fl_jacarey

Fixed typo on data width for writing to scratch memory.

Change 27974 on 2002/05/14 by vliu@vliu_r400_cnvliu100_win_cvd

Update.

Change 27956 on 2002/05/14 by jayw@MA_JAYW

added lower precision for majority of DeGamma table.
New dither documentation in blend

Change 27949 on 2002/05/14 by rbell@rbell_crayola_win_cvd

small fix

Change 27946 on 2002/05/14 by vliu@vliu_r400_cnvliu100_win_cvd

Update

Change 27945 on 2002/05/14 by rvelez@rvelez_r400_win_tor

Updated May 14

Change 27943 on 2002/05/14 by rbell@rbell_crayola_win_cvd

Added more tests for Milestone 1

Change 27926 on 2002/05/14 by imuskatb@imuskatb_r400_win_cnimuskatb

updated all crtc test to include DACA_EN
Updated crtc emu features doc
updated display_out to include crc computation
added one debug test for dispout

Change 27914 on 2002/05/14 by jacarey@fl_jacarey

Update Local Address Map

Change 27876 on 2002/05/14 by lseiler@lseiler_r400_win_marlboro

RB register and architecture overview

Change 27869 on 2002/05/14 by jacarey@fl_jacarey

Baseline CP Validation Plans

Change 27866 on 2002/05/14 by jacarey@fl_jacarey

Added CGM rtr signals back to the RBBM.

Change 27861 on 2002/05/14 by jacarey@fl_jacarey

Updated Scratch Memory Connections
Updated Me_Debug_data Connections

Change 27848 on 2002/05/14 by vliu@vliu_r400_cnvliu100_win_cvd

Update

Change 27835 on 2002/05/14 by efong@efong_r400_win_tor_doc

changed the h/v_total for 600 and setup some more worksheets for printing

Change 27832 on 2002/05/14 by jacarey@fl_jacarey

ME_Init Shader Base addresses are in Instructions, not DWORDs

Change 27824 on 2002/05/14 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 27823 on 2002/05/14 by efong@efong_r400_win_tor_doc

line buffer test description

Change 27822 on 2002/05/14 by gregs@gregs_r400_win_marlboro

update

Change 27818 on 2002/05/14 by vliu@vliu_r400_cnvliu100_win_cvd

Update

Change 27761 on 2002/05/13 by gabarca@gabarca_crayola_win_cvd

Updated features for milestone 1

Change 27752 on 2002/05/13 by rbell@rbell_crayola_win_cvd

Update

Change 27751 on 2002/05/13 by jayw@MA_JAYW

updated mc interface to eliminate access_subset and not the
offsetting of the access_address by the frame buffer offset for
non-external accesses.

Change 27729 on 2002/05/13 by jacarey@fl_jacarey

Changed Num_Constants to Max_Contexts in the ME_Init Packet.

Change 27717 on 2002/05/13 by llefebvr@llefebvre_laptop_r400

Changed CF opcodes, SQ->SP interface and SP->SQ constant index load interface.

Change 27714 on 2002/05/13 by jhoule@MA_JHOULE

TP corrections

Change 27713 on 2002/05/13 by georgev@ma_georgev

Some extra clarification.

Change 27707 on 2002/05/13 by georgev@ma_georgev

Added sections from verifcatin spec.

Change 27706 on 2002/05/13 by rvelez@rvelez_r400_win_tor

Updated 5/13

Change 27695 on 2002/05/13 by jacarey@fl_jacarey

Update Address Map
Top-Level Diagram Updates

Change 27679 on 2002/05/13 by jacarey@fl_jacarey

Update Local Addresses

Change 27673 on 2002/05/13 by jasif@jasif_r400_win_tor

Updated.

Change 27670 on 2002/05/13 by imuskatb@imuskatb_r400_win_cnimuskatb

Updated test to 4 active lines
updated crtc test desp

Change 27669 on 2002/05/13 by chwang@chwang_doc_r400_win_cvd

Added a few missing models.

Change 27665 on 2002/05/13 by vliu@vliu_r400_cnvliu100_win_cvd

Update

Change 27664 on 2002/05/13 by rbell@rbell_crayola_win_cvd

Graphics/Overlay tests for milestone 1

Change 27658 on 2002/05/13 by mdoggett@MA_MDOGGETT_LT

Added vertex fetch slide

Change 27657 on 2002/05/13 by jhoule@MA_JHOULE

Added EXP_ADJUST, pseudo-assembly for border color, and fast-path mention

Change 27654 on 2002/05/13 by efong@efong_r400_win_tor_doc

Updated

Change 27653 on 2002/05/13 by semara@semara_r400_win_tor

update the render secation

Change 27651 on 2002/05/13 by mdoggett@MA_MDOGGETT_LT

Merged TC slides in

Change 27650 on 2002/05/13 by tho@tho_r400_win

updated

Change 27648 on 2002/05/13 by mdoggett@MA_MDOGGETT_LT

First version of TC SW review

Change 27645 on 2002/05/13 by jhoule@MA_JHOULE

For merging with Michael

Change 27644 on 2002/05/13 by jacarey@fl_jacarey

Instruction Load packets support loading of the entire instruction
memory.

Change 27640 on 2002/05/13 by rbell@rbell_crayola_win_cvd

Added

Change 27634 on 2002/05/13 by jacarey@fl_jacarey

update cp spec.

Change 27623 on 2002/05/13 by chwang@chwang_doc_r400_win_cvd

Small update

Change 27621 on 2002/05/13 by csampayo@fl_csampayo_lt_r400

Updated VGT tests status

Change 27596 on 2002/05/13 by rbell@rbell_crayola_win_cvd

updated

Change 27592 on 2002/05/13 by ygiang@ygiang_r400_win_marlboro_p4

update

Change 27591 on 2002/05/13 by gregs@gregs_r400_sun_marlboro

new model.

Change 27589 on 2002/05/13 by fhsien@fhsien_r400_win_home_marlboro

Change little at home

Change 27588 on 2002/05/13 by smoss@smoss_crayola_win

update

Change 27536 on 2002/05/10 by ashishs@fl_ashishs_r400_win

update

Change 27525 on 2002/05/10 by fhsien@fhsien_r400_win_marlboro

Update with new info

Change 27523 on 2002/05/10 by csampayo@fl_csampayo_r400

Updated status for the following test:
r400vgt_dma_index_rectangle_list_01

Change 27510 on 2002/05/10 by frising@ma_frising

v.1.39
-4KB aligned for all textures
-Update data format table to be consistent with RB, add interpolate
and replicate YUV formats
-Move to Larry's convention for endian swapping
-Remove FETCH_VALID_ONLY in vertex instruction and replace with
MUST_BE_ONE
-Start to clean up usage tables. Lots to do still.

Change 27502 on 2002/05/10 by georgev@ma_georgev

First revision so that everyone else can get it in Perforce.

Change 27499 on 2002/05/10 by jasif@jasif_r400_win_tor

Updated dccif model owner.

Change 27488 on 2002/05/10 by rbell@rbell_crayola_win_cvd

Updated for milestone 1 GRP/OVL features

Change 27467 on 2002/05/10 by jacarey@fl_jacarey

Fix Typo

Change 27463 on 2002/05/10 by jhoule@MA_JHOULE

For SW review (unfinished)

Change 27455 on 2002/05/10 by gregs@gregs_r400_win_marlboro

update

Change 27439 on 2002/05/10 by jacarey@fl_jacarey

Update

Change 27431 on 2002/05/10 by jacarey@fl_jacarey

Instructions Added to Simplify Reading the Micro RAM
Fix Typo in the Pseudocode.

Change 27421 on 2002/05/10 by jacarey@fl_jacarey

Updates for ME_INIT Packet

Change 27415 on 2002/05/10 by jacarey@fl_jacarey

Update for Registers and Text Update to the CP Specification.

Change 27406 on 2002/05/10 by jacarey@fl_jacarey

Update to Internal Address Map (Comments Only)

Change 27386 on 2002/05/10 by imuskatb@imuskatb_r400_win_cnimuskatb

Updated crtc_contorller to handle boundary conditions
golden for the first 4 test of crtc
dispout updated7

Change 27385 on 2002/05/10 by jacarey@fl_jacarey

Checkpoint Updates to Specification
Update list of things to do.

Change 27373 on 2002/05/10 by omesh@ma_omesh

Added test cases for Destination Color Blending as derived from Source Color Blending
and also added notes on Source Alpha Blending test strategy.

Change 27329 on 2002/05/10 by jhoule@MA_JHOULE

Changed PREDICATE_CONDITION to PRED_CONDITION in PRED_SELECT
description.

Change 27325 on 2002/05/10 by llefebvr@llefebvre_laptop_r400

Presentation for the 5/13.

Change 27323 on 2002/05/10 by jhoule@MA_JHOULE

1.38
Instruction changes:
Added PRED_SELECT and PRED_CONDITION for the sequencer to manage pixel
masks.
Moved FETCH_VALID_ONLY to bit 19 in TFetch, and added it to VFetch.

Change 27322 on 2002/05/10 by jacarey@fl_jacarey

1. Remove CONST_PREFETCH Packet
2. Update LCC pseudocode for PFP

Change 27316 on 2002/05/10 by jacarey@fl_jacarey

Optional processing of Type-2 and Type-3 NOPs by ME.

Change 27314 on 2002/05/10 by rherrick@ma_rherrick_crayola

Not sure what was checked in for last revision since SAMBA was acting flaky on me... Rechecking in the changes

Change 27313 on 2002/05/10 by rherrick@ma_rherrick_crayola

Fixed ADDRESS documentation to match implementation...

Change 27302 on 2002/05/10 by smorein@smorein_r400

First rewrite of programing guide. Some parts still missing, will go in on monday.

Change 27264 on 2002/05/09 by jowang@jowang_R400_win

VF Control -- RGB

Change 27254 on 2002/05/09 by jacarey@fl_jacarey

STQ is now 114 deep.

Change 27251 on 2002/05/09 by jacarey@fl_jacarey

Update CNT register loading

Change 27241 on 2002/05/09 by jacarey@fl_jacarey

Update PM4 spec for me_init

Change 27237 on 2002/05/09 by jacarey@fl_jacarey

Updated Picture

Change 27233 on 2002/05/09 by grayc@grayc_r400_win

describes test bench for cp/rbbm

Change 27230 on 2002/05/09 by jasif@jasif_r400_win_tor

Vga emulator features and test schedule.

Change 27223 on 2002/05/09 by ashishs@fl_ashishs_r400_win

update

Change 27221 on 2002/05/09 by gregs@gregs_r400_win_marlboro

updated with Y25LF05 info.

Change 27214 on 2002/05/09 by omesh@ma_omesh

Added RBC Verification Plan (new file), currently under development, as well as a link to it from the main RB Verification Plan document.

Change 27208 on 2002/05/09 by chwang@chwang_doc_r400_win_cvd

Small update.

Change 27188 on 2002/05/09 by jacarey@fl_jacarey

Update

Change 27153 on 2002/05/09 by jacarey@fl_jacarey

Checkpoint Partial Update of CP Registers.

Change 27152 on 2002/05/09 by ygiang@ygiang_r400_win_marlboro_p4

update

Change 27127 on 2002/05/09 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 27064 on 2002/05/08 by nbarbier@nbarbier_r400_win_tor

Updated register test information.

Change 27053 on 2002/05/08 by gabarca@gabarca_crayola_win_cvd

submit

Change 27048 on 2002/05/08 by jacarey@fl_jacarey

Corrections for 8 Sub Blocks

Change 27046 on 2002/05/08 by jacarey@fl_jacarey

Checkpoint Specifications
1. ME_INIT Packet

2. Updates to CP Registers

... Still Not Done Yet ...

Change 27042 on 2002/05/08 by imuskatb@imuskatb_r400_win_cnimuskatb

Updated documents
fix latest lb, fix chunck_size
new crtc_test with golden

Change 27039 on 2002/05/08 by sallen@sallen_r400_win_marlboro

updates

Change 27038 on 2002/05/08 by rvelez@rvelez_r400_win_tor

Updated schedule for CRTC/DISPOUT

Change 27021 on 2002/05/08 by jimmylau@jimmylau_r400_win_tor

H sync and V sync polarities are set to different values for normal CRTC and VGA CRTC test cases.
Add a list of CRTC registers that are not tested in normal CRTC register test because they are either read-only or write only.

Change 27009 on 2002/05/08 by rvelez@rvelez_r400_win_tor

Updated May 8

Change 26998 on 2002/05/08 by jacarey@fl_jacarey

Add Booleans to Source Select

Change 26996 on 2002/05/08 by khabbari@khabbari_r400_win

lb test plan changes

Change 26991 on 2002/05/08 by tien@ma_spinach

Updated the tp area page.

Change 26984 on 2002/05/08 by rbagley@ma_rbagley_ltxp

Updates include a round of revision from Andi Skende, including updates to the shader pipe related material.

Change 26978 on 2002/05/08 by ygiang@ygiang_r400_win_marlboro_p4

v1

Change 26959 on 2002/05/08 by jacarey@fl_jacarey

Update Invalidate_State and Subblock_Prefetch packets
to support eight sub-blocks.

Change 26949 on 2002/05/08 by jacarey@fl_jacarey

Fix typo in rbbm_perf_cntl register.

Change 26934 on 2002/05/08 by tho@tho_r400_win

-name of document changed
-icon/cursor status updated

Change 26928 on 2002/05/08 by khabbari@khabbari_r400_win

added the start of line to lb/dcp interface doc

Change 26924 on 2002/05/08 by efong@efong_r400_win_tor_doc

moved this file

Change 26923 on 2002/05/08 by efong@efong_r400_win_tor_doc

moved to be with the design block.

Change 26922 on 2002/05/08 by nbarbier@nbarbier_r400_win_tor

Added register test for DCIO block.

Change 26920 on 2002/05/08 by tho@tho_r400_win

updated document w/ icon cursor status

Change 26919 on 2002/05/08 by efong@efong_r400_win_tor_doc

1st rev of the scaler feature list for emulator

Change 26901 on 2002/05/08 by jacarey@fl_jacarey

1. Set_State supports eight sub-blocks
2. Updates to Set_CF_Constant

Change 26886 on 2002/05/08 by jacarey@fl_jacarey

Update name of register that CP writes for the Booleans.

Change 26885 on 2002/05/08 by jacarey@fl_jacarey

1. Set_State:
a. Eight Sub-Blocks
b. Sub-Block Size Encoding = (n+1)*8

2. New Packet : Set_CF_Constant

Change 26884 on 2002/05/08 by jacarey@fl_jacarey

Sub-block Size Encoding Update

Change 26854 on 2002/05/07 by csampayo@fl_csampayo_r400

Updated VGT test status, as well as, limited other units test status to 99%

Change 26843 on 2002/05/07 by jowang@jowang_R400_win

Totally out of date.

Change 26842 on 2002/05/07 by imuskatb@imuskatb_r400_win_cnimuskatb

Test and Feature list. First pass

Change 26841 on 2002/05/07 by jowang@jowang_R400_win

Initial VF design.

Change 26839 on 2002/05/07 by jowang@jowang_R400_win

renamed to VF Block.vsd

Change 26835 on 2002/05/07 by jowang@jowang_R400_win

Initial VF Design

Change 26821 on 2002/05/07 by rvelez@rvelez_r400_win_tor

Split up DCCIF and DCCARB documents

Change 26819 on 2002/05/07 by gregs@gregs_r400_sun_marlboro

changed paths to chip directory + revision history + verified that it works.

Change 26808 on 2002/05/07 by jacarey@fl_jacarey

Moving Microcode Source

Change 26798 on 2002/05/07 by jacarey@fl_jacarey

Update Revision

Change 26795 on 2002/05/07 by jacarey@fl_jacarey

Added Type-0 Packet Support to Microcode.

Change 26792 on 2002/05/07 by jayw@MA_JAYW

added notebook

Change 26784 on 2002/05/07 by jayw@MA_JAYW

DeGamma increased precision to avoid
errors when temporarily over 1.0 due to
0.055 add.

Change 26779 on 2002/05/07 by rvelez@rvelez_r400_win_tor

Updated 5/7/02

Change 26762 on 2002/05/07 by gregs@gregs_r400_sun_marlboro

This is unchanged R300 file.

Change 26760 on 2002/05/07 by ygiang@ygiang_r400_win_marlboro_p4

updated: test list

Change 26759 on 2002/05/07 by bryans@bryans_crayola_doc

Updated as per status meeting

Change 26727 on 2002/05/07 by smoss@smoss_crayola_win

SU tests

Change 26721 on 2002/05/07 by llefebvr@llefebvre_laptop_r400

presentation for the SW meeting (draft)

Change 26711 on 2002/05/07 by ashishs@fl_ashishs_r400_win

update

Change 26701 on 2002/05/07 by rbell@rbell_crayola_win_cvd

First draft of DCP feature list

Change 26642 on 2002/05/06 by jimmylau@jimmylau_r400_win_tor

Add feature to start H and V counter at some even offset other than 0.
Add feature to choose whether to cutoff H sync A/B at when H blank ends.

Change 26638 on 2002/05/06 by vliu@vliu_r400_cnvliu100_win_cvd

Update

Change 26627 on 2002/05/06 by lkang@lkang_r400_win_tor

update on tristate h/vsync's in d1, D2 & D3

Change 26614 on 2002/05/06 by jacarey@fl_jacarey

Checkpoint Code

Change 26605 on 2002/05/06 by jacarey@fl_jacarey

Update to Micro Instructions

Change 26598 on 2002/05/06 by jacarey@fl_jacarey

Update to Access Jump Table in RAM

Change 26574 on 2002/05/06 by rvelez@rvelez_r400_win_tor

Updated May 6/02

Change 26555 on 2002/05/06 by mdoggett@MA_MDOGGETT_LT

Updates to TP, TPC, MH interfaces; L1 Tags; TCO diagram; TCD Padding; L2 cache pixel formats table; L2 cacheline format conversions table; 2d l2 one lyaer cacheline format

Change 26554 on 2002/05/06 by jacarey@fl_jacarey

MicroCode for Type1 packets.

Change 26532 on 2002/05/06 by lkang@lkang_r400_win_tor

Toronto milestone #1 schedule

Change 26529 on 2002/05/06 by jimmylau@jimmylau_r400_win_tor

Change registers DxCRTC_H/V_SYNC_A_END to 13 bits, due to added feature of testing H/V counter wrap around.
Update VGA vertical timing parameters to 11 bits.

Change 26525 on 2002/05/06 by jimmylau@jimmylau_r400_win_tor

Change register name from DxCRTC_H/V_COUNT_OFFSET to DxCRTC_H/V_SYNC_A_START.
Change test cases for H/V counter wrap around so that H/V_SYNC_A_START are even numbers,
since hardware does not support odd number H/V_SYNC_A_START.

Change 26510 on 2002/05/06 by tho@tho_r400_win

updated

Change 26506 on 2002/05/06 by vliu@vliu_r400_cnvliu100_win_cvd

Rename the file

Change 26505 on 2002/05/06 by jacarey@fl_jacarey

2D Indirect Buffer Invalidates Pointers instead
of the hardware doing this implicitly.

Change 26504 on 2002/05/06 by vliu@vliu_r400_cnvliu100_win_cvd

emulation / simulation feature list

Change 26500 on 2002/05/06 by nluu@nluu_r400_doclib_cnnb

- update

Change 26495 on 2002/05/06 by chwang@chwang_doc_r400_win_cvd

Update.

Change 26491 on 2002/05/06 by efong@efong_r400_win_tor_doc

Updated

Change 26487 on 2002/05/06 by jasif@jasif_r400_win_tor

Updated.

Change 26484 on 2002/05/06 by mmantor@mmantor_r400_win

update per comments from Doug

Change 26478 on 2002/05/06 by ygiang@ygiang_r400_win_marlboro_p4

added: more coments and tests to list

Change 26474 on 2002/05/06 by jacarey@fl_jacarey

Checkpoint Microcode Before CPP Updates

Change 26473 on 2002/05/06 by rbell@rbell_crayola_win_cvd

Added VeriBench User's Guide

Change 26470 on 2002/05/06 by imuskatb@imuskatb_r400_win_cnimuskatb

updated

Change 26465 on 2002/05/06 by rbell@rbell_crayola_win_cvd

updated

Change 26395 on 2002/05/03 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 26388 on 2002/05/03 by mmantor@mmantor_r400_win

added state module description

Change 26377 on 2002/05/03 by gregs@gregs_r400_win_marlboro

update.

Change 26371 on 2002/05/03 by scroce@scroce_r400_win_marlboro

update

Change 26370 on 2002/05/03 by jayw@MA_JAYW

friday night gamma fixed

Change 26364 on 2002/05/03 by rvelez@rvelez_r400_win_tor

Update May 3/02

Change 26354 on 2002/05/03 by hartogs@fl_hartogs

Fixed order of selections for VGT_OTUPUT_PATH_CNTL register.

Change 26332 on 2002/05/03 by jacarey@fl_jacarey

Checkpoint All Documents and Microcode.

Change 26326 on 2002/05/03 by georgev@ma_georgev

Changes for first primlib tests, changes to tg.cpp to support

ranges, SX spreadsheet and verification document, and a tg.h
that is out of date, but the good one is missing.

Change 26301 on 2002/05/03 by jacarey@fl_jacarey

1. Baseline Checklist for Post June 15th
2. Microcode Update
   a. Main Loop

Change 26292 on 2002/05/03 by gregs@gregs_r400_win_marlboro

added to verification section.

Change 26281 on 2002/05/03 by ygiang@ygiang_r400_win_marlboro_p4

updated: comments

Change 26252 on 2002/05/03 by jimmylau@jimmylau_r400_win_tor

removed test features : stereosync and dual CRTCs
add test features : H/V counter tests, CRTC register tests

Change 26250 on 2002/05/03 by gregs@gregs_r400_win_marlboro

spelling

Change 26242 on 2002/05/03 by jacarey@fl_jacarey

1. REPL command Added to OPER field.
2. Clarification that SUB* implies "SRC0 - SRC1"
3. MOV instruction is translated to:
      oper=shl;immed=0x00000000;

Change 26237 on 2002/05/03 by jacarey@fl_jacarey

Indicate input stage as a repeater.

Change 26229 on 2002/05/03 by ashishs@fl_ashishs_r400_win

update

Change 26224 on 2002/05/03 by bryans@bryans_crayola_doc

Renamed: R400 VideoIP DV Methodology -> VideoIP DV Methodology
Moved document to simulator/environment directory

Change 26205 on 2002/05/03 by jacarey@fl_jacarey

Microcode:

Changed jbit bool=<bit> @<address>;
to: jbit @<address>;bool=<bit>;

Change 26166 on 2002/05/02 by rvelez@rvelez_r400_win_tor

Split up DCCIF Interface Spec to separate DCCIF and DCCARB interfaces

Change 26165 on 2002/05/02 by rvelez@rvelez_r400_win_tor

Split up DCCIF and DCCARB interfaces into two documents

Change 26154 on 2002/05/02 by fhsien@fhsien_r400_win_marlboro

Update Depth tests

Change 26152 on 2002/05/02 by vliu@vliu_r400_cnvliu100_win_cvd

Modify the structure of the test_lib/src tree.

Change 26149 on 2002/05/02 by jayw@MA_JAYW

more fixes now starting monotinicity

Change 26138 on 2002/05/02 by smoss@smoss_crayola_win

SU tests and golds

Change 26118 on 2002/05/02 by gregs@gregs_r400_win_marlboro

changed default value of SUBSYS_ID ROM based strap.

Change 26111 on 2002/05/02 by efong@efong_r400_win_tor_doc

Initial revision

Change 26110 on 2002/05/02 by scroce@scroce_r400_win_marlboro

Added examples and a lot of info.

Change 26103 on 2002/05/02 by smoss@smoss_crayola_win

SU tests, golds

Change 26067 on 2002/05/02 by rvelez@rvelez_r400_win_tor

Updated 5/2

Change 26066 on 2002/05/02 by aashkar@fl_aashkar

Removed unused address

Change 26065 on 2002/05/02 by aashkar@fl_aashkar

fixed typo

Change 26057 on 2002/05/02 by llefebvr@llefebvre_laptop_r400

Modification on the Control flow instructions.

Change 26055 on 2002/05/02 by rherrick@ma_rherrick_crayola

Parser is working... Producing same results as Hard coded version... Documentation updated to reflect new byteEnable and Tag inclusion in ADDRESS directive...

Change 26054 on 2002/05/02 by ygiang@ygiang_r400_win_marlboro_p4

Added: Test list for SP

Change 26046 on 2002/05/02 by khabbari@khabbari_r400_win

small changes

Change 26037 on 2002/05/02 by aashkar@fl_aashkar

Added RT_ENABLE to address map

Change 26036 on 2002/05/02 by lseiler@lseiler_r400_win_marlboro

Replaced by the Autoreg generated register description

Change 26035 on 2002/05/02 by jacarey@fl_jacarey

Type 1 Packet Corrections

Change 26033 on 2002/05/02 by bryans@bryans_crayola_doc

Update to include emulator diagram/flow with reify

Change 26030 on 2002/05/02 by aashkar@fl_aashkar

Added missing Address for Write Confirm ID FIFO

Change 26028 on 2002/05/02 by jayw@MA_JAYW

added example and fixed some gamma bugs

Change 26025 on 2002/05/02 by jacarey@fl_jacarey

Microcode
1. Type1 Packet
2. NOP Packet
3. Reserved Bit Check Routine
4. Invalid Opcode Check Routine

Change 26013 on 2002/05/02 by kcorrell@kcorrell_r400_docs_marlboro

updated TC interface, DC interface and buffering, added auxiliary logic section. Modified address translation to include HDP_FB_START and ROM_BASE.

Change 25975 on 2002/05/02 by jacarey@fl_jacarey

Clarified use of the texture constants.

Change 25958 on 2002/05/02 by jacarey@fl_jacarey

Partial Update to Top-Level Diagram

Change 25957 on 2002/05/02 by jacarey@fl_jacarey

Partial updates for supporting new 2D packets.

Change 25955 on 2002/05/02 by jacarey@fl_jacarey

Removed signals that no longer exist:
RBBM_CP_3d_idle
CP_RBBM_2d_idle

Change 25917 on 2002/05/01 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 25912 on 2002/05/01 by nbarbier@nbarbier_r400_win_tor

Made changes following test plan review for first milestone.

Change 25896 on 2002/05/01 by lkang@lkang_r400_win_tor

update on 5/1

Change 25850 on 2002/05/01 by jayw@MA_JAYW

Only Gamma monotinicity left.

Change 25824 on 2002/05/01 by jacarey@fl_jacarey

Update signal names for the memory hub power management.

Change 25816 on 2002/05/01 by jhoule@MA_JHOULE

0.9.9-0.9.10:
Inserted complete size restriction from .xls file.
Wrapping/Clamping description.
TP->TC interface suggestion.

Change 25806 on 2002/05/01 by jacarey@fl_jacarey

Update

Change 25805 on 2002/05/01 by gregs@gregs_r400_win_marlboro

added another open issue.

Change 25804 on 2002/05/01 by jacarey@fl_jacarey

Micro Engine Details

Change 25802 on 2002/05/01 by gregs@gregs_r400_win_marlboro

added another ROM: Y25LF05

Change 25778 on 2002/05/01 by jacarey@fl_jacarey

Renamed MH_CP_coherency_busy to MH_RBBM_coherency_busy

Change 25772 on 2002/05/01 by jacarey@fl_jacarey

Update Docuents with ME details.

Change 25770 on 2002/05/01 by gregs@gregs_r400_win_marlboro

added default value of MEM_AP_SIZE as an open issue.

Change 25763 on 2002/05/01 by jacarey@fl_jacarey

Added RT Enable Flag

Change 25719 on 2002/05/01 by gabarca@gabarca_crayola_win_cvd

updated

Change 25694 on 2002/05/01 by jacarey@fl_jacarey

Checkpoint Code

Change 25685 on 2002/05/01 by gregs@gregs_r400_win_marlboro

added comments on TMDS pwr/gnd requirements.

Change 25652 on 2002/04/30 by jayw@MA_JAYW

minor update in progress

Change 25629 on 2002/04/30 by jacarey@fl_jacarey

Update Micro Engine Local Addresses

Change 25628 on 2002/04/30 by jacarey@fl_jacarey

Details to Micro Engine
Local Addresses for Micro Engine

Change 25624 on 2002/04/30 by scroce@scroce_r400_win_marlboro

added PLI info

Change 25622 on 2002/04/30 by gabarca@gabarca_crayola_win_cvd

updated

Change 25596 on 2002/04/30 by hartogs@fl_hartogs

Version 0.93
Added conversion method "VGT_GRP_FIX_1_23_TO_FLOAT" to the list of specifiable conversion methods in the register VGT_GROUP_VECT_0_FMT_CNTL. This conversion method is, at the time of this revision, used solely by the tessellation engine to convert interpolation weights from fixed point 1.23 format to 32-bit IEEE floating point.
Added "RETAIN_ORDER" and "RETAIN_QUADS" fields to the register VGT_GROUP_PRIM_TYPE.
Added description of how the prim type information flows through the VGT to section 5.2.1.

Change 25586 on 2002/04/30 by efong@efong_r400_win_tor_doc

updated with crtc_en signal

Change 25581 on 2002/04/30 by ashishs@fl_ashishs_r400_win

update

Change 25570 on 2002/04/30 by ashishs@fl_ashishs_r400_win

update

Change 25561 on 2002/04/30 by ashishs@fl_ashishs_r400_win

update

Change 25558 on 2002/04/30 by gabarca@gabarca_crayola_win_cvd

Finished establisshing features and sub-features to be tested, developed the VGA HDP section, almost finished the VGA HDP test table spreadsheet

Change 25550 on 2002/04/30 by jacarey@fl_jacarey

Checkpoint Micro Engine Documentation

Change 25547 on 2002/04/30 by lkang@lkang_r400_win_tor

update on 4/30

Change 25540 on 2002/04/30 by ashishs@fl_ashishs_r400_win

update

Change 25521 on 2002/04/30 by ashishs@fl_ashishs_r400_win

update

Change 25508 on 2002/04/30 by jacarey@fl_jacarey

Update for Local Micro Engine Addresses

Change 25458 on 2002/04/29 by gabarca@gabarca_crayola_win_cvd

updated

Change 25457 on 2002/04/29 by gabarca@gabarca_crayola_win_cvd

updated

Change 25450 on 2002/04/29 by csampayo@fl_csampayo_r400

Update status for the following VGT test
r400vgt_index_size_01

Change 25418 on 2002/04/29 by sbagshaw@sbagshaw

VIP status for chip integration as of April 23rd, 2002

Change 25415 on 2002/04/29 by jayw@MA_JAYW

must run

Change 25413 on 2002/04/29 by jacarey@fl_jacarey

Checkpoint Specifications

Change 25386 on 2002/04/29 by jasif@jasif_r400_win_tor

Updated forcible signals section to use prefix "force" for all forcible signal names.

Change 25372 on 2002/04/29 by gabarca@gabarca_crayola_win_cvd

updated

Change 25354 on 2002/04/29 by lkang@lkang_r400_win_tor

update on 4/29

Change 25346 on 2002/04/29 by fhsien@fhsien_r400_win_marlboro

Only change the format of the template for RB verification

Change 25339 on 2002/04/29 by jasif@jasif_r400_win_tor

Updated

Change 25335 on 2002/04/29 by rbell@rbell_crayola_win_cvd

updated

Change 25334 on 2002/04/29 by gabarca@gabarca_crayola_win_cvd

Put in VGA HDP

Change 25326 on 2002/04/29 by nluu@nluu_r400_doclib_cnnb

- update

Change 25324 on 2002/04/29 by mmantor@mmantor_r400_win

updated spec for PA_SC_su interface changes
updated sc.v and created tb directories

Change 25305 on 2002/04/29 by efong@efong_r400_win_tor_doc

Updated

Change 25304 on 2002/04/29 by chwang@chwang_doc_r400_win_cvd

Update.

Change 25303 on 2002/04/29 by askende@andi_r400_docs

modifications to RB-SX interfaces

Change 25296 on 2002/04/29 by omesh@ma_omesh

Just changed the heading/title of document. Nothing else.

Change 25295 on 2002/04/29 by gabarca@gabarca_crayola_win_cvd

Cahnged name

Change 25280 on 2002/04/29 by jacarey@fl_jacarey

No special logic in the RBBM for detecting IDCT register transactions. Any logic needed will be in the CP's micro engine.

Change 25278 on 2002/04/29 by gabarca@gabarca_crayola_win_cvd

Cahnged name

Change 25277 on 2002/04/29 by tho@tho_r400_win

updated

Change 25275 on 2002/04/29 by gabarca@gabarca_crayola_win_cvd

Update

Change 25273 on 2002/04/29 by vliu@vliu_r400_cnvliu100_win_cvd

Update

Change 25268 on 2002/04/29 by jacarey@fl_jacarey

Update Pre-Fetch Parser

Change 25197 on 2002/04/26 by csampayo@fl_csampayo_lt_r400

Added status for the following tests:
r400vgt_index_offset_01
r400vgt_index_offset_02
r400vgt_index_offset_03

Change 25193 on 2002/04/26 by csampayo@fl_csampayo_lt_r400

Resubmit previous

Change 25192 on 2002/04/26 by csampayo@fl_csampayo_lt_r400

Added index offset to VGT section

Change 25187 on 2002/04/26 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 25148 on 2002/04/26 by ashishs@fl_ashishs_r400_win

just checked out

Change 25145 on 2002/04/26 by ashishs@fl_ashishs_r400_win

update

Change 25135 on 2002/04/26 by scroce@scroce_r400_win_marlboro

Added some new features and modified some current features

Change 25134 on 2002/04/26 by jayw@MA_JAYW

minor tweak to the exponent table, adding one to exponent
added +1.0 bybass in front of degamma.

Change 25105 on 2002/04/26 by jacarey@fl_jacarey

1. Remove GEN_INT_* references
2. No Shared Registers -- Update read return bus
3. Added MASTER_INT_SIGNAL register

Change 25075 on 2002/04/26 by jacarey@fl_jacarey

Update Documents

Change 25056 on 2002/04/25 by gregs@gregs_r400_win_marlboro

update

Change 25055 on 2002/04/25 by gregs@gregs_r400_win_marlboro

updated number of analog sipply pins + descriptions.

Change 25046 on 2002/04/25 by jayw@MA_JAYW

added MS gamma blech

Change 25020 on 2002/04/25 by ashishs@fl_ashishs_r400_win

update

Change 25014 on 2002/04/25 by ashishs@fl_ashishs_r400_win

update

Change 24968 on 2002/04/25 by georgev@ma_georgev

First Rev.

Change 24961 on 2002/04/25 by ctaylor@fl_ctaylor_r400_win_marlboro

Spec Updates

Change 24941 on 2002/04/25 by jayw@MA_JAYW

fixed a missing zero in DeGamma example and improved formatting of the result.

Change 24919 on 2002/04/25 by rherrick@ma_rherrick_crayola

Updated spec to reflect two Texture Cache interface clients

Change 24910 on 2002/04/25 by csampayo@fl_csampayo_r400

Updated the schedule and the status for the following tests:
r400vgt_index_min_max_01
r400vgt_index_min_max_02

Change 24905 on 2002/04/25 by jacarey@fl_jacarey

Fixed typo of force_mismatch

Change 24904 on 2002/04/25 by lkang@lkang_r400_win_tor

update on 4/24

Change 24902 on 2002/04/25 by lkang@lkang_r400_win_tor

update on 4/24

Change 24901 on 2002/04/25 by lkang@lkang_r400_win_tor

update after spec review

Change 24899 on 2002/04/25 by rherrick@ma_rherrick_crayola

Improve documentation to reflect implementation (including data-spec definition)

Change 24888 on 2002/04/25 by jayw@jayw_r400_win_home

Added example of DeGamma to blend doc and added exponent table documentation

Change 24832 on 2002/04/24 by bryans@bryans_crayola_doc

Added general flow diagrams (more to come)

Change 24827 on 2002/04/24 by jayw@MA_JAYW

Added graph around DEBUGX

Change 24822 on 2002/04/24 by jayw@MA_JAYW

made debugging a particular x easier

Change 24810 on 2002/04/24 by jayw@MA_JAYW

added debug lines at the end

Change 24807 on 2002/04/24 by jayw@MA_JAYW

fixed with no implicit leading one and exponent
bias of +15 as per Paul Vella's request.

Change 24796 on 2002/04/24 by gregs@gregs_r400_win_marlboro

SPLL registers update.

Change 24713 on 2002/04/24 by georgev@ma_georgev

Opened for edit.

Change 24712 on 2002/04/24 by smoss@smoss_crayola_win

updated su tests

Change 24707 on 2002/04/24 by bryans@bryans_crayola_doc

Update directory tree (parts_lib) for virage/testchip_vid subdirs

Change 24676 on 2002/04/24 by jacarey@fl_jacarey

Added Read Request Ports to the MIU

Change 24675 on 2002/04/24 by jacarey@fl_jacarey

Powerpoint slides from 2D PM4 Review

Change 24674 on 2002/04/24 by jacarey@fl_jacarey

Checkpoint All Documents.

Change 24673 on 2002/04/24 by jayw@jayw_r400_win_home

missing files added

Change 24639 on 2002/04/23 by csampayo@fl_csampayo_lt_r400

Updated status for the following tests:
r400vgt_reuse_depth_triangle_list_01
r400vgt_reuse_depth_line_list_01

Change 24615 on 2002/04/23 by jimmylau@jimmylau_r400_win_tor

move CRTC test plan documents under doc_lib/design/blocks/dc/crtc

Change 24604 on 2002/04/23 by gregs@gregs_r400_win_marlboro

update

Change 24574 on 2002/04/23 by rbagley@ma_rbagley_ltxp

Some updates (far from complete) to the syntax. More important, the doc is reorganized.

Change 24542 on 2002/04/23 by lkang@lkang_r400_win_tor

update on 4/23

Change 24540 on 2002/04/23 by tien@ma_spinach

Updated TP area estimate

Change 24536 on 2002/04/23 by lkang@lkang_r400_win_tor

update on 4/23

Change 24519 on 2002/04/23 by jayw@MA_JAYW

changed tag between MC and RB to 9 bits.

Change 24516 on 2002/04/23 by jayw@MA_JAYW

fixed perl bug, more usefull X's found now.

Change 24489 on 2002/04/23 by jimmylau@jimmylau_r400_win_tor

Add crtc_en signal in the SCL-CRTC interface specs.

Change 24484 on 2002/04/23 by frising@ma_frising

24 bit floating point format is E4M20 not E5M19. No version bump.

Change 24462 on 2002/04/23 by nbarbier@nbarbier_r400_win_tor

Initial Revision.

Change 24445 on 2002/04/23 by jacarey@fl_jacarey

Misc. Updates to 2D Appendix
Reset "Valid" flags on 3D-to-2D Transitions in PFP.

Change 24416 on 2002/04/22 by csampayo@fl_csampayo_r400

Updated status for the test
r400vgt_reuse_depth_point_list_01

Change 24406 on 2002/04/22 by bryans@bryans_crayola_doc

Update per status meeting

Change 24387 on 2002/04/22 by mmantor@mmantor_r400_win

update to sc dumps and added last tile to walker out dump in sc_walker. Also updated the standslone exe and the scan converter spec for the new sc=>sq interface

Change 24385 on 2002/04/22 by gregs@gregs_r400_win_marlboro

update

Change 24353 on 2002/04/22 by lkang@lkang_r400_win_tor

vga integration

Change 24352 on 2002/04/22 by lkang@lkang_r400_win_tor

display integration plan

Change 24266 on 2002/04/22 by chwang@chwang_doc_r400_win_cvd

Addition of BIF Model test plan.

Change 24243 on 2002/04/22 by gabarca@gabarca_crayola_win_cvd

VGAREG_VGA_MEMORY_BASE_ADDRESS [31:25] Represents the start of the 32 Meg aligned, 32 Meg sized area of memory where the VGA HDP and rendering reads and writes.

Change 24234 on 2002/04/22 by chwang@chwang_doc_r400_win_cvd

Updated with some implementation details.

Change 24230 on 2002/04/22 by vliu@vliu_r400_cnvliu100_win_cvd

Update

Change 24218 on 2002/04/22 by jasif@jasif_r400_win_tor

Updated.

Change 24209 on 2002/04/22 by efong@efong_r400_win_tor_doc

Updated

Change 24195 on 2002/04/22 by chwang@chwang_doc_r400_win_cvd

Update.

Change 24191 on 2002/04/22 by tho@tho_r400_win

updated

Change 24187 on 2002/04/22 by imuskatb@imuskatb_r400_win_cnimuskatb

Updated

Change 24173 on 2002/04/22 by rbell@rbell_crayola_win_cvd

updated

Change 24170 on 2002/04/22 by gregs@gregs_r400_win_marlboro

removed ROM sram area, as we are not going to have any memory in ROM.

Change 24082 on 2002/04/19 by csampayo@fl_csampayo_r400

Updated status for following tests
r400vgt_reuse_index_triangle_list_01
r400vgt_reuse_index_line_list_01
r400vgt_reuse_index_point_list_01

Change 24050 on 2002/04/19 by jacarey@fl_jacarey

Remove constant flag from MIU logic.

Change 24049 on 2002/04/19 by pmitchel@pmitchel_r400_win_marlboro

add

Change 24024 on 2002/04/19 by jacarey@fl_jacarey

PM4 Spec Updates for Write Confirmation
Microcode Updates for write confirmation.

Change 24008 on 2002/04/19 by gregs@gregs_r400_win_marlboro

update.

Change 23984 on 2002/04/19 by georgev@ma_georgev

Master and sub documents for verification plan

Change 23961 on 2002/04/19 by smoss@smoss_crayola_win

added culling

Change 23958 on 2002/04/19 by bryans@bryans_crayola_doc

Add macros:

CHIP_EMU_ARCHS_CONFS
CHIP_SIM_ARCHS_CONFS
<test>_emu_archs_confs
<test>_sim_archs_confs

to support specifying pairs of arch/conf within the Makefile.

Note: CHIPARCH= and CHIPCONF= can be used on the command line to override

Change 23957 on 2002/04/19 by llefebvr@llefebvre_laptop_r400

The new control flow scheme is now included in v2.0 of the sequencer spec.

Change 23946 on 2002/04/19 by llefebvr@llefebvre_laptop_r400

Last version of the spec with the old control flow scheme

Change 23940 on 2002/04/19 by jacarey@fl_jacarey

Updated signal name to CP_CG_2d_mode.

Change 23939 on 2002/04/19 by jacarey@fl_jacarey

Added CG_CP_2d_mode to real-time stream connections document.

Change 23926 on 2002/04/19 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 23893 on 2002/04/18 by csampayo@fl_csampayo_r400

Updated status for following tests:
r400vgt_ext2int_index_line_loop_01
r400vgt_ext2int_index_triangle_strip_01
r400vgt_ext2int_index_quad_strip_01
r400vgt_ext2int_index_triangle_fan_01
r400vgt_ext2int_index_quad_list_01
r400vgt_ext2int_index_polygon_01
r400vgt_ext2int_index_triangle_list_01

Change 23851 on 2002/04/18 by gabarca@gabarca_crayola_win_cvd

Added working version of VGA test plan:

Change 23836 on 2002/04/18 by jacarey@fl_jacarey

1. Add TBD to RBBM spec for interrupts
2. Update packets for "write confirmation"

Change 23835 on 2002/04/18 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 23767 on 2002/04/18 by jacarey@fl_jacarey

Update Connections to MIU

Change 23726 on 2002/04/17 by csampayo@fl_csampayo_r400

Updated status for the following tests:
r400vgt_ext2int_index_points_01
r400vgt_ext2int_index_line_list_01
r400vgt_ext2int_index_line_strip_01

Change 23722 on 2002/04/17 by askende@andi_r400_docs

new rev. 03 checked in.

Change 23718 on 2002/04/17 by abeaudin@abeaudin_r400_win_marlboro

update transaction engine

Change 23714 on 2002/04/17 by vromaker@MA_VIC_P4

updated SQ memory sizes

Change 23661 on 2002/04/17 by efong@efong_r400_win_tor_doc

changed from VGA_DISP_d1_rotate_90_deg to VGA_DISP_d2_rotate_90_deg

Change 23635 on 2002/04/17 by efong@efong_r400_win_tor_doc

Updated due to review as well as new pin (CRTC_SCL_display_read_request_dis)

Change 23617 on 2002/04/17 by jacarey@fl_jacarey

Checkpoint Micro Engine Diagrams
Microcode includes the main loop (Packet Dispatch)

Change 23597 on 2002/04/17 by gregs@gregs_r400_win_marlboro

updated register fields.

Change 23590 on 2002/04/17 by jayw@MA_JAYW

MaxErr, better error range checking with quatization effects.

Change 23573 on 2002/04/17 by efong@efong_r400_win_tor_doc

Updated due to review as well as new spec still have to update the reify file stuff

Change 23566 on 2002/04/17 by gregs@gregs_r400_win_marlboro

updated MEM_AP_SIZE encodings + strap readback table.

Change 23535 on 2002/04/16 by csampayo@fl_csampayo_r400

Updated r400vgt_provoking_vtx_all_01 test status

Change 23529 on 2002/04/16 by jacarey@fl_jacarey

Checkpoint Microcode -- A lot of Updates

Change 23525 on 2002/04/16 by smoss@smoss_crayola_win

updated su stuff

Change 23395 on 2002/04/15 by whui@whui_r400_win_tor

Update open issue

Change 23370 on 2002/04/15 by jacarey@fl_jacarey

Baseline Micro Engine Address Map
Check-Point Microcode

ME_INIT packet for PM4 Spec
Register Updates for Cp Spec

Change 23345 on 2002/04/15 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 23309 on 2002/04/15 by jacarey@fl_jacarey

Update FIFO sizes and fix placement of the "rcd" signal.

Change 23307 on 2002/04/15 by jacarey@fl_jacarey

Document ati_rbbm_intf usage in cp's rbiu

Change 23302 on 2002/04/15 by efong@efong_r400_win_tor_doc

Update the date

Change 23300 on 2002/04/15 by imuskatb@imuskatb_r400_win_cnimuskatb

updated

Change 23296 on 2002/04/15 by jasif@jasif_r400_win_tor

Updated.

Change 23289 on 2002/04/15 by chwang@chwang_doc_r400_win_cvd

Update.

Change 23287 on 2002/04/15 by dwong@cndwong2

xDCT test plan version 0.1

Only sections 3 (3.1..3.3) and 7 (7.2) contain valid information

Change 23282 on 2002/04/15 by nluu@nluu_r400_doclib_cnnb

- update

Change 23262 on 2002/04/15 by vliu@vliu_r400_cnvliu100_win_cvd

Update

Change 23259 on 2002/04/15 by tho@tho_r400_win

updated

Change 23242 on 2002/04/15 by efong@efong_r400_win_tor_doc

Updated

Change 23237 on 2002/04/15 by gregs@gregs_r400_win_marlboro

update.

Change 23234 on 2002/04/15 by gregs@gregs_r400_win_marlboro

added table of strap output signals

Change 23233 on 2002/04/15 by rbell@rbell_crayola_win_cvd

updated

Change 23227 on 2002/04/15 by jacarey@fl_jacarey

Details of Idle and Clean for wait until logic.

Change 23203 on 2002/04/14 by jiezhou@jiezhou_r400_win_tor

1)Change icon read return data format (P0 start from LSB)
to keep the same format with R300.

2)Change cursor read return data format (P0 start from LSB)

Change 23188 on 2002/04/13 by jimmylau@jimmylau_r400_win_tor

Initial test plan for CRTC

Change 23164 on 2002/04/12 by lkang@lkang_r400_win_tor

overall updating

Change 23163 on 2002/04/12 by csampayo@fl_csampayo_r400

Added description of r400vgt_index_source_switch_01

Change 23124 on 2002/04/12 by gregs@gregs_r400_win_marlboro

removed apad_strength3_3V straps.

Change 23112 on 2002/04/12 by khabbari@khabbari_r400_win

frame controller changes reflecting the review

Change 23075 on 2002/04/12 by jayw@MA_JAYW

Friday night check-in

Change 23074 on 2002/04/12 by jayw@MA_JAYW

Initial Gamma and DeGamma derivation code.

Change 23038 on 2002/04/12 by grayc@grayc_r400_win

described generic fifo

Change 23036 on 2002/04/12 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 22989 on 2002/04/12 by gregs@gregs_r400_win_marlboro

1. added registers for fuse box for "bad pipes"
2. added support for new devices: W45B512, W45B012

Change 22966 on 2002/04/11 by jiezhou@jiezhou_r400_win_tor

no changing

Change 22965 on 2002/04/11 by jiezhou@jiezhou_r400_win_tor

Update sub-block
"graphic and overlay alpha blending" -> include GRPH_EN and OVL_EN control
"Window Controller" -> Add more description of surface updating infomation
Updated state machine for graphic and overlay request
Remove "Y shift 1 bit" at 32 bpp digital output case and
overlay half resolution case
Section 5.1 -> Add signal desciption of 1/8 of H-total from CRTC as we don't have
interface spec of DCP-CRTC

"LUT" -> update register name and autofill data value for 8-bit mode.

Change 22920 on 2002/04/11 by rbell@rbell_crayola_win_cvd

Updates

Change 22919 on 2002/04/11 by jimmylau@jimmylau_r400_win_tor

Major update on CRTC specs :
remove AUXWIN and related registers
remove VGA timing and mode registers. Get signals from VGA block instead.
major updates on register fields
updates on CRTC interface signals and subblock interface signals
major updates on section 6 : Data Processing Algorithms

Change 22903 on 2002/04/11 by gregs@gregs_r400_win_marlboro

typo error fixed.

Change 22894 on 2002/04/11 by mkelly@fl_mkelly_r400_win_laptop

update

Change 22890 on 2002/04/11 by gregs@gregs_r400_win_marlboro

added ROM straps and fuse box.

Change 22886 on 2002/04/11 by nbarbier@nbarbier_r400_win_tor

Updated signal names.

Change 22883 on 2002/04/11 by nbarbier@nbarbier_r400_win_tor

Initial revision.

Change 22878 on 2002/04/11 by jacarey@fl_jacarey

Update addresses for incremental update tests for constants and instruction memory.

Change 22868 on 2002/04/11 by csampayo@fl_csampayo_r400

Added status of following tests:
r400su_parallel_orientation_all_01
r400su_parallel_orientation_all_02

Change 22867 on 2002/04/11 by jacarey@fl_jacarey

Clear the VS and PS valid flags in the PFP if the driver
writes the shader instruction code directly.

Change 22866 on 2002/04/11 by semara@semara_r400_win_tor

adding the pixelwriter

Change 22843 on 2002/04/11 by rbell@rbell_crayola_win_cvd

Renamed doc file name

Change 22831 on 2002/04/11 by pmitchel@MA_PAULM_P4

move to proper place

Change 22818 on 2002/04/11 by georgev@MA_YVALCOUR

Old version.

Change 22814 on 2002/04/11 by georgev@MA_YVALCOUR

Change property again.

Change 22805 on 2002/04/11 by georgev@MA_YVALCOUR

Changed edit protection to lock.

Change 22800 on 2002/04/11 by gregs@gregs_r400_win_marlboro

minor corrections

Change 22799 on 2002/04/11 by gregs@gregs_r400_win_marlboro

removed the BUSY signals table.

Change 22761 on 2002/04/10 by csampayo@fl_csampayo_r400

Added status of the following tests:
r400vgt_dma_index_points_01
r400vgt_dma_index_line_list_01
r400vgt_dma_index_line_strip_01
r400vgt_dma_index_line_loop_01
r400vgt_immed_index_points_01
r400vgt_immed_index_line_list_01
r400vgt_immed_index_line_strip_01
r400vgt_immed_index_line_loop_01

Change 22755 on 2002/04/10 by jiezhou@jiezhou_r400_win_tor

Change "DxOVL_MATRIX_TRANSFORM_EN"
"DxOVL_PWL_TRANSFORM_EN"
"DxCOLOR_SUBSAMPLE_CRCB_MODE[1:0]"
"DxCOLOR_MATRIX_TRANSFORM_EN"
from attribute of "double buffered" to "single buffered"

Add "DxGRPH_FLIP_MODE" to Graphic group

Move "DxGRPH_ENABLE" to Graphic group
"DxOVL_ENABLE" to Overlay group

Some change in LUT group:

Move "DxGRPH_LUT_SEL" to Graphic group
Change "DxLUT_AUTOFILL" to "DC_LUT_AUTOFILL"
Change "DC_LUT_MODE" to "DC_LUT_RW_MODE" with single buffered
attribute

Change "DC_LUT_READb_WRITE_SELECT" to "DC_LUT_RW_SEL"
Change "DC_LUT_INDEX[7:0]" to "DC_LUT_RW_INDEX[7:0]"
Change "DC_LUT_WRITE_EN_MASK[5:0]" from "double buffered" to "single buffered"

Change 22709 on 2002/04/10 by llefebvr@llefebvre_laptop_r400

Control flow proposal updated

Change 22708 on 2002/04/10 by jacarey@fl_jacarey

Remove mention of single ring mode for the instruction memory management.

Change 22700 on 2002/04/10 by jacarey@fl_jacarey

Removal of Dual Ring Mode for Instruction Memory

Change 22683 on 2002/04/10 by gregs@gregs_r400_win_marlboro

static screen update

Change 22680 on 2002/04/10 by georgev@MA_YVALCOUR

First revision

Change 22679 on 2002/04/10 by georgev@MA_YVALCOUR

No Change.

Change 22669 on 2002/04/10 by khabbari@khabbari_r400_win

frame controller for scaler released

Change 22628 on 2002/04/10 by jacarey@fl_jacarey

Updates:
1. Synchronization
2. Performance Counters

Change 22594 on 2002/04/10 by bryans@bryans_crayola_doc

Update

Change 22589 on 2002/04/10 by jacarey@fl_jacarey

Update to Specifications for Synchronization

Change 22577 on 2002/04/09 by beiwang@bei_mcspec

Updated to reflect deletion of Address Return Bus and updates to RB bus interface

Change 22544 on 2002/04/09 by sallen@sallen_r400_win_marlboro

Move ferret and rename spec

Change 22510 on 2002/04/09 by mkelly@fl_mkelly_r400_win_laptop

Scissor rect tests...

Change 22507 on 2002/04/09 by khabbari@khabbari_r400_win

removed DCP_LB_P1_EN from lb/dcp interface

Change 22491 on 2002/04/09 by khabbari@khabbari_r400_win

added subsample modes signal to both lb/dcp and lb/scl interface docs

Change 22483 on 2002/04/09 by hartogs@fl_hartogs

Updated draw initiator comments for "INDEX_SIZE" and "SOURCE_SELECT" fields based on behaviors learned from the emulator.

Change 22476 on 2002/04/09 by frising@ma_frising

v.1.37
-make surface alignment for 2d/3d linear textures be 4KB.

Change 22473 on 2002/04/09 by abeaudin@abeaudin_r400_win_marlboro

documentation for registers and the gfx transaction engine

Change 22450 on 2002/04/09 by georgev@MA_YVALCOUR

First revision.

Change 22443 on 2002/04/09 by kcorrell@kcorrell_r400_docs_marlboro

Next update

Change 22393 on 2002/04/08 by khabbari@khabbari_r400_win

lb/scaler and lb/dcp interface update

Change 22359 on 2002/04/08 by jasif@jasif_r400_win_tor

Updated.

Change 22348 on 2002/04/08 by bryans@bryans_crayola_doc

Fix dc configuration to include VIP

Change 22319 on 2002/04/08 by bbloemer@ma-jasonh

Added DRAM bank pin names, separate from address pins.

Change 22317 on 2002/04/08 by mkelly@fl_mkelly_r400_win_laptop

Simple scissor rectangle test...

Change 22316 on 2002/04/08 by llefebvr@llefebvre_laptop_r400

The control flow proposal

Change 22313 on 2002/04/08 by jacarey@fl_jacarey

Updates for synchronization

Change 22302 on 2002/04/08 by gregs@gregs_r400_win_marlboro

clock names

Change 22293 on 2002/04/08 by mkelly@fl_mkelly_r400_win_laptop

SC scissor rectangle test

Change 22291 on 2002/04/08 by vliu@vliu_r400_cnvliu100_win_cvd

Update

Change 22281 on 2002/04/08 by gregs@gregs_r400_win_marlboro

clock names and spelling

Change 22277 on 2002/04/08 by jowang@jowang_R400_win

For first milestone

Change 22276 on 2002/04/08 by nluu@nluu_r400_doclib_cnnb

- update

Change 22275 on 2002/04/08 by jacarey@fl_jacarey

Checkpoint Specifications

Change 22273 on 2002/04/08 by chwang@chwang_doc_r400_win_cvd

Update.

Change 22264 on 2002/04/08 by efong@efong_r400_win_tor_doc

Updated

Change 22261 on 2002/04/08 by tho@tho_r400_win

updated

Change 22260 on 2002/04/08 by gregs@gregs_r400_win_marlboro

updated clock names

Change 22259 on 2002/04/08 by imuskatb@imuskatb_r400_win_cnimuskatb

updated

Change 22253 on 2002/04/08 by abeaudin@abeaudin_r400_win_marlboro

removed old schedules

Change 22243 on 2002/04/08 by jasif@jasif_r400_win_tor

Updated.

Change 22241 on 2002/04/08 by jacarey@fl_jacarey

Updates to diagram.

Change 22239 on 2002/04/08 by rbell@rbell_crayola_win_cvd

Small update for config example

Change 22237 on 2002/04/08 by rbell@rbell_crayola_win_cvd

updated

Change 22231 on 2002/04/08 by jacarey@fl_jacarey

1. Update to PFP Pseudocode
2. Diagram for CP-to-Driver Synchronization
3. Checkpoint Microcode
4. Checkpoint RBBM Specification

Change 22162 on 2002/04/05 by jiezhou@jiezhou_r400_win_tor

update some field name to be consistent with verilog register file.

Change 22152 on 2002/04/05 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 22149 on 2002/04/05 by jacarey@fl_jacarey

Small clarification for event_timestamp_write packet.

Change 22143 on 2002/04/05 by jacarey@fl_jacarey

Updated EVENT_TIMESTAMP_WRITE PM4 packet.
Updated count fields for PFP-to-ME intermediate packets.

Change 22135 on 2002/04/05 by chwang@chwang_doc_r400_win_cvd

Minor update about model arguments.

Change 22130 on 2002/04/05 by chwang@chwang_doc_r400_win_cvd

Added updated Reify spec.

Change 22120 on 2002/04/05 by nluu@nluu_r400_doclib_cnnb

- add more details to the implementation section

Change 22101 on 2002/04/05 by chwang@chwang_doc_r400_win_cvd

Update to the Coding Standard.

Change 22094 on 2002/04/05 by jacarey@fl_jacarey

Renamed CP_RBBM_rt_idle to CP_RBBM_rt_busy

Change 22079 on 2002/04/05 by lkang@lkang_r400_win_tor

display device power management update

Change 22078 on 2002/04/05 by semara@semara_r400_win_tor

adding VGA_DISP_d1/2_rotate_90_deg signals

Change 22068 on 2002/04/05 by semara@semara_r400_win_tor

Updating the interface to use display width/height to interface to the scalar.
adjust the vertical parameters indexes from [9:0] to [10:0].

Change 22067 on 2002/04/05 by jacarey@fl_jacarey

Remove Discrete signals for RT streams from CP Specification.

Change 22066 on 2002/04/05 by jacarey@fl_jacarey

Update to reflect "start of frame" counters for Display Engines.

Change 22065 on 2002/04/05 by jacarey@fl_jacarey

Document for Discrete RT stream connections instead
of placing this information in the CP Specification.

Change 22056 on 2002/04/05 by llefebvr@llefebvre_laptop_r400

The new control flow proposal.

Change 22037 on 2002/04/05 by bryans@bryans_crayola_doc

Add dc_disp configuration

Change 21976 on 2002/04/04 by nluu@nluu_r400_doclib_cnnb

- add implementation specs to DCCIF Model.

Change 21973 on 2002/04/04 by csampayo@fl_csampayo_r400

Updated VGT section with tests written

Change 21964 on 2002/04/04 by gregs@gregs_r400_win_marlboro

update

Change 21919 on 2002/04/04 by nluu@nluu_r400_doclib_cnnb

- add bit width to signals in diagrams

Change 21906 on 2002/04/04 by nluu@nluu_r400_doclib_cnnb

- mark don't cares on all relevant signals

Change 21904 on 2002/04/04 by khabbari@khabbari_r400_win

the lb test plan for phase1

Change 21894 on 2002/04/04 by jimmylau@jimmylau_r400_win_tor

Change SC-CRTC interface specs. Add signals to disable display requests for display1 and display2.

Change 21890 on 2002/04/04 by bryans@bryans_crayola_doc

Brief doc on environment variables and standard makefile defines

Change 21870 on 2002/04/04 by gregs@gregs_r400_win_marlboro

added input registers for RBBM_active and MH_active and pm_en.

Change 21859 on 2002/04/04 by gregs@gregs_r400_win_marlboro

moved ROM based straps table

Change 21853 on 2002/04/04 by gregs@gregs_r400_win_marlboro

reset -> srst

Change 21844 on 2002/04/04 by bhankins@fl_bhankins_r400_win

Initial checkin

Change 21831 on 2002/04/04 by gregs@gregs_r400_win_marlboro

verilog examples of clock gating

Change 21816 on 2002/04/04 by gregs@gregs_r400_win_marlboro

sclk gating example files.

Change 21798 on 2002/04/04 by jacarey@fl_jacarey

Miscellaneous Corrections to PM4 Specification.

Change 21771 on 2002/04/03 by jiezhou@jiezhou_r400_win_tor

small updating

Change 21770 on 2002/04/03 by csampayo@fl_csampayo_r400

PA Validation Tests Tracking Document

Change 21733 on 2002/04/03 by nluu@nluu_r400_doclib_cnnb

- change title in properties

Change 21719 on 2002/04/03 by nluu@nluu_r400_doclib_cnnb

- add more test cases

Change 21712 on 2002/04/03 by nluu@nluu_r400_doclib_cnnb

- add a no-reset test case

- change to revision 1.0

Change 21710 on 2002/04/03 by nluu@nluu_r400_doclib_cnnb

- Add implementation specs to rbbmif model doc

Change 21650 on 2002/04/03 by hartogs@fl_hartogs

changed VGT_SQ_end_of_vector to VGT_SQ_end_of_vtx_vect.
changed VGT_PA_clip_p_start_vector to VGT_PA_clip_p_new_vtx_vect.
fixed stride and shift entries for vector 0 in Major Mode 0 table.
Updated the "Active Issues" list.
Added the SWAP_MODE field to the DMA_SIZE register.
Changed the implied value of bit 2 of the VGT_MH_ad interface from a zero to a one. This indicates that the VGT is always doing 256-bit requests.

Change 21638 on 2002/04/03 by jacarey@fl_jacarey

Updated Invalidate_State packet to swap VS and PS bit order

Change 21634 on 2002/04/03 by abeaudin@abeaudin_r400_win_marlboro

cleaned up documentaion

Change 21620 on 2002/04/03 by jacarey@fl_jacarey

Clarified that the MH has a slip FIFO

Change 21619 on 2002/04/03 by gregs@gregs_r400_win_marlboro

Updated MH section

Change 21589 on 2002/04/03 by jacarey@fl_jacarey

Added memclk active feedback to MIU.

Change 21586 on 2002/04/03 by jacarey@fl_jacarey

Added CP_MH_reqmem signal to the CP.

Change 21533 on 2002/04/03 by jacarey@fl_jacarey

Update to All Documents

Change 21517 on 2002/04/02 by csampayo@fl_csampayo_r400

For documenting PA validation environment/processes

Change 21492 on 2002/04/02 by kcorrell@kcorrell_r400_docs_marlboro updated interfaces

Change 21467 on 2002/04/02 by bryans@bryans_crayola_doc

more updates (open issues)

Change 21423 on 2002/04/02 by jasif@jasif_r400_win_tor

Updated.

Change 21396 on 2002/04/02 by bryans@bryans_crayola_doc

Update status based on Lili's changes

Change 21382 on 2002/04/02 by efong@efong_r400_win_tor_doc

prefixed all the forciable signals with force_

Change 21381 on 2002/04/02 by efong@efong_r400_win_tor_doc

prefixed the forciable signals with force_.

Change 21380 on 2002/04/02 by efong@efong_r400_win_tor_doc

changed the names of the forciable signals to be prefixed with force_.
changed the bus_timeout to be 100 from 50 because rbbmif will timeout after 68 cycles.
fixed up the table with the commands and removed 0x200 from all the write cycles
put down some extra limitations of the model
put in a small section in the rd file section for timeout

Change 21377 on 2002/04/02 by jayw@MA_JAYW

added RB_MH_queuecount
and removed tile_done from RBs.

Change 21366 on 2002/04/02 by jayw@MA_JAYW

complement for RB_HW_design_spec.doc
area estimates, etc.

Change 21365 on 2002/04/02 by jayw@MA_JAYW

MH_RB_queuecount_external is now MH_RB_queuecount

Change 21349 on 2002/04/02 by rbell@rbell_crayola_win_cvd

updated

Change 21345 on 2002/04/02 by jacarey@fl_jacarey

Interface Updates for CP.

Change 21314 on 2002/04/01 by jiezhou@jiezhou_r400_win_tor

Initial release for dcp test plan

Change 21308 on 2002/04/01 by bryans@bryans_crayola_doc

Post status meeting update

Change 21290 on 2002/04/01 by jasif@jasif_r400_win_tor

Modified description of timing diagram.

Change 21277 on 2002/04/01 by gregs@gregs_r400_win_marlboro

added pin_strap_DRAM_select

Change 21253 on 2002/04/01 by lkang@lkang_r400_win_tor

Move the file from devel/doc_lib to doc_lib

Change 21251 on 2002/04/01 by jasif@jasif_r400_win_tor

Updated unterface signal names to match released dccif.v top level.

Change 21222 on 2002/04/01 by bryans@bryans_crayola_doc

Update status

Change 21220 on 2002/04/01 by lkang@lkang_r400_win_tor

milestones

Change 21216 on 2002/04/01 by jasif@jasif_r400_win_tor

Added assumptions section.

Change 21207 on 2002/04/01 by nluu@nluu_r400_doclib_cnnb

- update status

Change 21206 on 2002/04/01 by jasif@jasif_r400_win_tor

Updated Error Section.

Change 21204 on 2002/04/01 by efong@efong_r400_win_tor_doc

Updated

Change 21201 on 2002/04/01 by jasif@jasif_r400_win_tor

Updated.

Change 21198 on 2002/04/01 by chwang@chwang_doc_r400_win_cvd

Update.

Change 21195 on 2002/04/01 by dclifton@dclifton_r400

Moved perspective correction of I and J to VTE. Updated the RBIU interface to very latest concept of state variable storage (subject to change again).

Change 21191 on 2002/04/01 by vliu@vliu_r400_cnvliu100_win_cvd

Update

Change 21189 on 2002/04/01 by jasif@jasif_r400_win_tor

Updated

Change 21182 on 2002/04/01 by jayw@MA_JAYW

Monday morning checkin.
Some tile cache drawings in Visio.

Change 21177 on 2002/04/01 by tho@tho_r400_win

updated

Change 21161 on 2002/04/01 by rbell@rbell_crayola_win_cvd

updated

Change 21134 on 2002/03/31 by beiwang@bei_mcspec

Deleted text descriptions on Address return bus to be consistent w/ the bus interface tables

Change 21122 on 2002/03/31 by semara@semara_r400_win_tor

updating the address gen section

Change 20941 on 2002/03/28 by bbloemer@ma-jasonh

Updated dc queue text.

Change 20934 on 2002/03/28 by hartogs@fl_hartogs

More clean-up. From this point on, revisions to the spec will be detailed in the revision section.

Change 20912 on 2002/03/28 by jacarey@fl_jacarey

Added soft reset for VGT to RBBM.
Multiple Updates to CP specs for 2D.

Change 20890 on 2002/03/28 by rherrick@ma_rherrick_crayola

Fixed Table Of Contents...

Change 20883 on 2002/03/28 by gregs@gregs_r400_win_marlboro

added block to block timing

Change 20880 on 2002/03/28 by rherrick@ma_rherrick_crayola

Revamped specification to include relative transaction methods (instead of directed vs pattern methods)... Also added client synchronization...

Change 20861 on 2002/03/28 by rbell@rbell_crayola_win_cvd

updated

Change 20859 on 2002/03/28 by whui@whui_r400_win_tor

updated section 8: power management

Change 20808 on 2002/03/28 by hartogs@fl_hartogs

Added and deleted several state registers.
Updated interface tables.

Change 20799 on 2002/03/28 by jacarey@fl_jacarey

Added clock gating information to the CP Specification.

Change 20769 on 2002/03/27 by mmantor@mmantor_r400_win

updated for interface integration changes

Change 20739 on 2002/03/27 by jasif@jasif_r400_win_tor

Added force prefix for signal names forcible from tests.

Change 20721 on 2002/03/27 by jayw@MA_JAYW

removed bogus offset x and offset y from detail bus.
added event bit to coarse bus.

Change 20720 on 2002/03/27 by nluu@nluu_r400_doclib_cnnb

- initial revision of DCCIF Model test plan

Change 20718 on 2002/03/27 by gregs@gregs_r400_win_marlboro

Updated with the global core clock idea.

Change 20715 on 2002/03/27 by nluu@nluu_r400_doclib_cnnb

- correct links - model documents were re-organized

Change 20711 on 2002/03/27 by bryans@bryans_crayola_doc

Update the arch/conf spec for simulation

Change 20710 on 2002/03/27 by jacarey@fl_jacarey

Update RBBM Interfaces

Change 20699 on 2002/03/27 by jacarey@fl_jacarey

Update Spec for Interfaces

Change 20692 on 2002/03/27 by jayw@MA_JAYW

fixed a couple of minor signal names.

Change 20678 on 2002/03/27 by askende@andi_r400_docs

a new rev of the shader export spec.

Change 20664 on 2002/03/27 by jacarey@fl_jacarey

Interface Updates

Change 20641 on 2002/03/27 by jacarey@fl_jacarey

Added "Read Combine Disable" to diagrams.

Change 20632 on 2002/03/27 by dglen@dglen_r400_dell

Doing major update. Still not complete.

Change 20606 on 2002/03/26 by rvelez@rvelez_r400_win_tor

Latest DCCIF changes - fixed typos in docs

Change 20601 on 2002/03/26 by jacarey@fl_jacarey

Update CP Interfaces

Change 20589 on 2002/03/26 by khabbari@khabbari_r400_win

tvout/dispout interface doc released

Change 20584 on 2002/03/26 by jacarey@fl_jacarey

Sweeping Update to the Interface Signal List.

Change 20580 on 2002/03/26 by jayw@MA_JAYW

Missing file, disappeared mysteriously

Change 20578 on 2002/03/26 by jayw@MA_JAYW

Released to match rb.v and rc.v

Change 20574 on 2002/03/26 by jowang@jowang_R400_win

Action Items 02/07

Change 20573 on 2002/03/26 by jayw@MA_JAYW

Several bus name changes.
Added RB->RC system context bus.

Change 20571 on 2002/03/26 by mpersaud@mpersaud_r400_win_tor

Changes for Feb 28 Design Review

Change 20560 on 2002/03/26 by jacarey@fl_jacarey

Interface Update for VIP, BIF reviews

Change 20531 on 2002/03/26 by jasif@jasif_r400_win_tor

Updated.

Change 20524 on 2002/03/26 by nluu@nluu_r400_doclib_cnnb

- update

Change 20513 on 2002/03/26 by dwong@cndwong2

R400 xDCT design review presentation material

Change 20512 on 2002/03/26 by jasif@jasif_r400_win_tor

Initial revision.

Change 20471 on 2002/03/26 by rthambim@rthambim_r400_win_tor

Added skid buffer structure diagram.

Change 20444 on 2002/03/25 by rthambim@rthambim_r400_win_tor

Updated write combining info.

Change 20442 on 2002/03/25 by rthambim@rthambim_r400_win_tor

Added skid buffer timing diagram.

Change 20426 on 2002/03/25 by askende@andi_r400_docs

a new rev (0.2)

Change 20403 on 2002/03/25 by jacarey@fl_jacarey

Clarify the fix-to-float is not normalized.

Change 20375 on 2002/03/25 by jayw@MA_JAYW

Weekend updates

Change 20364 on 2002/03/25 by jiezhou@jiezhou_r400_win_tor

small updating

Change 20357 on 2002/03/25 by semara@semara_r400_win_tor

rev 0.6

Change 20347 on 2002/03/25 by vliu@vliu_r400_cnvliu100_win_cvd

Update

Change 20339 on 2002/03/25 by jacarey@fl_jacarey

Checkpoint Diagram and Micro Code.

Change 20335 on 2002/03/25 by efong@efong_r400_win_tor_doc

updated

Change 20332 on 2002/03/25 by chwang@chwang_doc_r400_win_cvd

Update.

Change 20320 on 2002/03/25 by llefebvr@llefebvre_laptop_r400

Upated the interfaces and added an exporting rule section.

Change 20316 on 2002/03/25 by nluu@nluu_r400_doclib_cnnb

- update

Change 20313 on 2002/03/25 by jasif@jasif_r400_win_tor

Updated.

Change 20309 on 2002/03/25 by tho@tho_r400_win

updated

Change 20308 on 2002/03/25 by rbell@rbell_crayola_win_cvd

updates

Change 20303 on 2002/03/25 by nluu@nluu_r400_doclib_cnnb

- update

Change 20302 on 2002/03/25 by imuskatb@imuskatb_dv_win_cvd

update

Change 20294 on 2002/03/25 by jiezhou@jiezhou_r400_win_tor

combine DCP_Controller.doc into Display_Composite_Pipe.doc

Change 20293 on 2002/03/25 by rbell@rbell_crayola_win_cvd

changes

Change 20292 on 2002/03/25 by jiezhou@jiezhou_r400_win_tor

update

Change 20291 on 2002/03/25 by jiezhou@jiezhou_r400_win_tor major release to support architecture change:

1) add graphic/overlay alpha blending feature
2) add overlay matrix transformation block
3) add overlay gamma correction block
4) add 512 offset to icon expander
5) add 512 offset to cursor expander
6) add "pick-up" odd pixels in Sub-sampline block
7) some precision issues along the data path
8) remove supporting for liear mode
9) remove supporting for one pipe enabled at tiled-mode
10) combine DCP_Controller.doc into Display_Composite_Pipe.doc
11) DCP_register_list.doc separate from Display_Composite_Pipe.doc

Change 20289 on 2002/03/25 by jacarey@fl_jacarey

clarified the brush offset value format

Change 20288 on 2002/03/25 by jacarey@fl_jacarey

Update cp_interrupt packet to help the micro engine design.

Change 20287 on 2002/03/25 by jacarey@fl_jacarey

Fix Typos in the number formats

Change 20286 on 2002/03/25 by jacarey@fl_jacarey

Clarification on Packet Restrictions

Change 20269 on 2002/03/24 by rthambim@rthambim_r400_win_tor

Updated ROM cycle info.

Change 20268 on 2002/03/24 by rthambim@rthambim_r400_win_tor

Updated write combining info.

Change 20267 on 2002/03/24 by rthambim@rthambim_r400_win_tor

Fixed syntax.

Change 20237 on 2002/03/22 by jiezhou@jiezhou_r400_win_tor

Major update for DCP register list

Change 20234 on 2002/03/22 by rthambim@rthambim_r400_win_tor

Included strap table, updated review feedback.

Change 20219 on 2002/03/22 by askende@andi_r400_docs

first cut of the document

Change 20215 on 2002/03/22 by jacarey@fl_jacarey

Update Interface list

Change 20199 on 2002/03/22 by llefebvr@llefebvre_laptop_r400

Some minor changes to the SQ interfaces.

Change 20194 on 2002/03/22 by jacarey@fl_jacarey

Update Interfaces

Change 20192 on 2002/03/22 by mzhu@mzhu_r400_win_tor

remove support for single DCP pipeline

Change 20189 on 2002/03/22 by gregs@gregs_r400_win_marlboro

things copied from r300 and not yet modified - in red color.

Change 20187 on 2002/03/22 by gregs@gregs_r400_win_marlboro

Initial version - notes on pins, pad sizes, die and package sizes.

Change 20157 on 2002/03/22 by jacarey@fl_jacarey

Baseline CP Microcode
Typos in PM4 Spec

Change 20144 on 2002/03/22 by frising@ma_frising

No version change.
-added in bit field definitions for texture width/height/depth that we keep forgetting to add :)

Change 20142 on 2002/03/22 by mmantor@mmantor_r400_win

updated interface tables to represent actual planned hardware interfaces

Change 20131 on 2002/03/22 by dclifton@dclifton_r400

Updated drawings and interface to Scan Converter

Change 20130 on 2002/03/22 by jacarey@fl_jacarey

Miscellaneous corrections to references to VGT registers

Change 20129 on 2002/03/22 by jacarey@fl_jacarey

Rename vtg_index_offset to vgt_indx_offset

Change 20124 on 2002/03/22 by jowang@jowang_R400_win

added signals which are passed upstream from CRTC

Change 20114 on 2002/03/22 by jayw@MA_JAYW

Largely formatting.
Revamped MC interface and added RBBM.

Change 20102 on 2002/03/22 by jacarey@fl_jacarey

Update Draw_Indx and Viz_Query Pseudocode.

Change 20098 on 2002/03/22 by jhoule@MA_JHOULE

1.36:
Changed DST_SEL_* names from XYZW to SRC_X, SRC_Y, SRC_Z, and SRC_W.
Equivalent in SRC_SEL* to GRP_X, GPR_Y, and GPR_W.
Consistent capitalized first letter in fields.
Enable/Disable FORCE_BC_W_TO_MAX now Enabled/Disabled.

Change 20097 on 2002/03/22 by gregs@gregs_r400_win_marlboro

minor mod.

Change 20094 on 2002/03/22 by jowang@jowang_R400_win

reviewed and corrected description

Change 20092 on 2002/03/22 by jhoule@MA_JHOULE

Rev 0.9.8:
Updated SQ and SP interface from respective specs.

Change 20086 on 2002/03/22 by gregs@gregs_r400_win_marlboro

Updated memory interface.

Change 20058 on 2002/03/21 by abeaudin@abeaudin_r400_win_marlboro

new emulator documentation for adding blocks

Change 20018 on 2002/03/21 by jacarey@fl_jacarey

Clarifications on Set_state usage.
2D Updates.

Change 20015 on 2002/03/21 by jacarey@fl_jacarey

Removed Erroneous Note

Change 19995 on 2002/03/21 by bryans@bryans_crayola_doc

updated

Change 19979 on 2002/03/21 by bryans@bryans_crayola_doc

Realign goals

Change 19960 on 2002/03/21 by jacarey@fl_jacarey

Clarification of C0 and C1 constant loading from CP for 2D

Change 19914 on 2002/03/21 by rherrick@ma_rherrick_crayola

First Release for general consumption/review... Changes include:

More completed requirements documentation (in the form of Configuration Spec File Documentation)

Configuration File in a loose BNF format

Change 19865 on 2002/03/20 by nbarbier@nbarbier_r400_win_tor

Initial Release

Change 19858 on 2002/03/20 by efong@efong_r400_win_tor_doc

Updated with new BIF-VGA requirements

Change 19843 on 2002/03/20 by abeaudin@abeaudin_r400_win_marlboro

new documentation for emulator

Change 19820 on 2002/03/20 by jacarey@fl_jacarey

Updated Registers Per .blk file and Emulator

Change 19728 on 2002/03/20 by gregs@gregs_r400_win_marlboro

Initial version - copied (and re-arranged) from R300.

Change 19718 on 2002/03/20 by jacarey@fl_jacarey

Fix equation for load_constant_context packet.

Change 19705 on 2002/03/20 by efong@efong_r400_win_tor_doc

Updated revision stuff

Change 19704 on 2002/03/20 by efong@efong_r400_win_tor_doc

removed all the power stuff.

Change 19656 on 2002/03/19 by jacarey@fl_jacarey

Clarification for auto-generation of indices.

Change 19650 on 2002/03/19 by jacarey@fl_jacarey

Update to 2D Register Definitions.

Change 19631 on 2002/03/19 by jacarey@fl_jacarey

Update interface signals to RCIU.

Change 19629 on 2002/03/19 by jacarey@fl_jacarey

Fix typos on interface signals from CP and HI.

Change 19619 on 2002/03/19 by jacarey@fl_jacarey

Checkpoint Spec
1. PFP Constant coherency
2. Register Updates

Change 19595 on 2002/03/19 by jacarey@fl_jacarey

Update POLYLINE information.
Update line brush processing.

Change 19590 on 2002/03/19 by scroce@scroce_r400_win_marlboro

Updated Table of Contents

Change 19568 on 2002/03/19 by jacarey@fl_jacarey

Updated Brush_Offset plan for POLYLINE packet.

Change 19520 on 2002/03/18 by jiezhou@jiezhou_r400_win_tor

update

Change 19506 on 2002/03/18 by jiezhou@jiezhou_r400_win_tor

feature change to add alpha blending

Change 19503 on 2002/03/18 by llefebvr@llefebvre_laptop_r400

Changed the interfaces to reflect the fact that the PCs are now in the SX blocks

Change 19496 on 2002/03/18 by paulv@MA_PVELLA

Minor update to MHS section, including the DBR block diagram.

Change 19492 on 2002/03/18 by fhsien@fhsien_r400_win_marlboro

del for yung

Change 19485 on 2002/03/18 by kcorrell@kcorrell_r400_docs_marlboro

added files describing how mh treats chips addresses, minor tweaks to text

Change 19482 on 2002/03/18 by jacarey@fl_jacarey

Checkpoint CP Specification
* Partial Register Updates
  - BIOS Scratch Registers
  - Push Register Apertures
  - Updates to Register Text

Change 19478 on 2002/03/18 by jasif@jasif_r400_win_tor

Updated.

Change 19437 on 2002/03/18 by efong@efong_r400_win_tor_doc

Updated to new crtc spec. Also changed the DE to read_request

Change 19436 on 2002/03/18 by jacarey@fl_jacarey

Update Renderer Backend registers that are written by the CP
during the GUI Master Control handler. This is based on e-mail
from Larry Seiler.

Change 19424 on 2002/03/18 by vliu@vliu_r400_cnvliu100_win_cvd

Update

Change 19419 on 2002/03/18 by frising@ma_frising

Initial check-in.

Change 19417 on 2002/03/18 by efong@efong_r400_win_tor_doc

Updated

Change 19415 on 2002/03/18 by jimmylau@jimmylau_r400_win_tor

Update interface signal list based on CRTC block specs.

Change 19413 on 2002/03/18 by nluu@nluu_r400_doclib_cnnb

- update

Change 19409 on 2002/03/18 by chwang@chwang_doc_r400_win_cvd

Update.

Change 19406 on 2002/03/18 by jacarey@fl_jacarey

Update to wait_until conditions per e-mail from D. Glen on
1st and 2nd display/overlay controllers.

Change 19405 on 2002/03/18 by jasif@jasif_r400_win_tor

Updated.

Change 19380 on 2002/03/18 by jacarey@fl_jacarey

Add INDEX_OFFSET to the 3D Draw Packets.

Change 19369 on 2002/03/18 by tho@tho_r400_win

updated

Change 19346 on 2002/03/17 by jacarey@fl_jacarey

Illustration for 2D Surface Coherency

Change 19345 on 2002/03/17 by jacarey@fl_jacarey

Updates from Reviews

Change 19240 on 2002/03/15 by rbell@rbell_crayola_win_cvd

updates

Change 19217 on 2002/03/15 by semara@semara_r400_win_tor

update

Change 19216 on 2002/03/15 by rherrick@ma_rherrick_crayola

More completion of first pass spec...

Change 19201 on 2002/03/15 by scroce@scroce_r400_win_marlboro

Added info about alias file creation

Change 19156 on 2002/03/15 by frising@ma_frising

no version change
-fixed very minor typo on DATA_FORMAT page pointed out by Larry.

Change 19094 on 2002/03/15 by rherrick@ma_rherrick_crayola

More progress made... Still not complete... Redefined window setting directives...
Worked on first pass of Pattern Reads

Change 19057 on 2002/03/14 by lkang@lkang_r400_win_tor

updated section 6 on power state management

Change 19056 on 2002/03/14 by semara@semara_r400_win_tor

update section 10

Change 19051 on 2002/03/14 by frising@ma_frising

v.1.35
-moved some fields around in texture constants to faciliate 2D.
-add per quad and per pixel register lod to texture instruction.
-moved SIGNED_RF_MODE_ALL up one bit in vertex instruction.
-cleaned up a few comments.

Change 19034 on 2002/03/14 by nbarbier@nbarbier_r400_win_tor

Updated Interface

Change 19002 on 2002/03/14 by rherrick@ma_rherrick_crayola

Yet another deposit along the way... Not ready for review yet...

Change 18959 on 2002/03/14 by lseiler@ma_lseiler

Memory Formats: minor updates

Change 18958 on 2002/03/14 by lseiler@ma_lseiler

Render Backend minor updates

Change 18909 on 2002/03/14 by lkang@lkang_r400_win_tor

updated power management on pclk domain.

Change 18875 on 2002/03/14 by rherrick@nashua_rherrick_crayola

Progress on Spec.. No milestone...

Change 18834 on 2002/03/13 by nluu@nluu_r400_doclib_cnnb

- test plan for RBBMIF model

Change 18697 on 2002/03/13 by nluu@nluu_r400_doclib_cnnb

- correct comment about timeout and shared-registers

Change 18690 on 2002/03/13 by rvelez@rvelez_r400_win_tor

Top Level DC Spec

Change 18682 on 2002/03/13 by scroce@scroce_r400_win_marlboro

Quick comments

Change 18669 on 2002/03/13 by jacarey@fl_jacarey

Baseline Overview Presentations
Checkpoint PM4 Specification

Change 18660 on 2002/03/13 by rherrick@ma_rherrick_crayola

Documentation for MC/MH Test Environment

Change 18629 on 2002/03/12 by jasif@jasif_r400_win_tor

Updated arbitration between read and write requests. Added open issue of skid buffers.

Change 18616 on 2002/03/12 by nluu@nluu_r400_doclib_cnnb

- forgot to delete one doc in previous submit

Change 18607 on 2002/03/12 by khabbari@khabbari_r400_win

added fields to the register section of LB arch doc

Change 18594 on 2002/03/12 by nluu@nluu_r400_doclib_cnnb

- re-org, rename files to all lowercase

Change 18582 on 2002/03/12 by efong@efong_r400_win_tor_doc

forgot to change some of the names from HSYNC to HSYNCA/B and VSYNC to VSYNCA/B

Change 18564 on 2002/03/12 by khabbari@khabbari_r400_win

EOL removed from lb_dcp interface doc

Change 18552 on 2002/03/12 by jhoule@MA_JHOULE

0.9.7:
Added mipmap packing scheme description (with images).
Moved Addressing in a Varia section.
Changed some sub-block names.

Change 18486 on 2002/03/11 by dglen@dglen_r400_dell

Major update for first time in far too long.

Change 18483 on 2002/03/11 by semara@semara_r400_win_tor

rev 0.3

Change 18457 on 2002/03/11 by bbloemer@ma-jasonh

Updated clock naming convention.

Change 18454 on 2002/03/11 by nluu@nluu_r400_doclib_cnnb

- Legacy => legacy

Change 18450 on 2002/03/11 by nluu@nluu_r400_doclib_cnnb

- re-organize model docs.

Change 18437 on 2002/03/11 by jacarey@fl_jacarey

Updated Surface Coherence Equations for Lines and Fix Problems

Change 18421 on 2002/03/11 by jacarey@fl_jacarey

Miscellaneous Updates and Corrections

Change 18415 on 2002/03/11 by wlawless@wlawless

Revision 0.8 has updates to the color block cache, and its tag descriptions... Also a RB full data flow diagram and description was added.

Change 18385 on 2002/03/11 by jasif@jasif_r400_win_tor

Updated

Change 18379 on 2002/03/11 by rbell@rbell_crayola_win_cvd

update

Change 18376 on 2002/03/11 by khabbari@khabbari_r400_win

added new signals to the lb_dcp interface doc

Change 18357 on 2002/03/11 by jimmylau@jimmylau_r400_win_tor

Finalize CRTC naming convention to CRTC1/CRTC2
Remove the notation of internal/external overscan in CRTC. Overscan width is determined by scaler only.
Change the definition of ending timing parameters to be exclusive.
Add power management signals to CRTC-Scaler interface.

Change 18356 on 2002/03/11 by nluu@nluu_r400_doclib_cnnb

- update

Change 18355 on 2002/03/11 by chwang@chwang_doc_r400_win_cvd

Update

Change 18343 on 2002/03/11 by efong@efong_r400_win_tor_doc

Updated

Change 18342 on 2002/03/11 by tho@tho_r400_win

update

Change 18341 on 2002/03/11 by rthambim@rthambim_r400_win_tor

Fixed syntax.

Change 18331 on 2002/03/11 by jacarey@fl_jacarey

R400 Memory Map Diagram

Change 18323 on 2002/03/11 by jacarey@fl_jacarey

Implicit Sync Only in CF Pipe

Change 18322 on 2002/03/11 by jacarey@fl_jacarey

Updates from Comments
Update Surface Coherency Intersect Equation

Change 18320 on 2002/03/11 by jacarey@fl_jacarey

CP Overview
Updates to RBBM Specification

Change 18318 on 2002/03/10 by rthambim@rthambim_r400_win_tor

Added skid buffer timing diagram.

Change 18317 on 2002/03/10 by rthambim@rthambim_r400_win_tor

Fixed naming convention in timing diagram.

Change 18308 on 2002/03/10 by rthambim@rthambim_r400_win_tor

Fixed the block diagram.

Change 18305 on 2002/03/10 by rthambim@rthambim_r400_win_tor

Added fifo info, fixed slave implementation diagram.

Change 18292 on 2002/03/09 by frising@ma_frising

v.1.34
-introduced arbitrary_filter field. A few other feilds had to be juggled in the process.

Change 18284 on 2002/03/08 by paulv@MA_PVELLA

Forgot to update table of contents, etc. Fixed.

Change 18283 on 2002/03/08 by paulv@MA_PVELLA

Updated MHS portion of spec with latest specifications/fixes, including block diagrams.

Change 18280 on 2002/03/08 by rthambim@rthambim_r400_win_tor

Initial revision.

Change 18263 on 2002/03/08 by jacarey@fl_jacarey

Update RBBM Top-Level Diagram

Change 18240 on 2002/03/08 by rthambim@rthambim_r400_win_tor

Modified the RBBM-BIF timing diagram, address width, RBBMIF naming.

Change 18216 on 2002/03/08 by jacarey@fl_jacarey

Miscellaneous Corrections

Change 18211 on 2002/03/08 by jacarey@fl_jacarey

Updates for Reviews

Change 18181 on 2002/03/08 by bryans@bryans_crayola_doc

Updated status with dccif change

Change 18152 on 2002/03/08 by jacarey@fl_jacarey

Miscellaneous Updates

Change 18140 on 2002/03/08 by scroce@scroce_r400_win_marlboro

Updated for MKTREE changes

Change 18128 on 2002/03/08 by khabbari@khabbari_r400_win

small fix in line buffer arch

Change 18125 on 2002/03/08 by jacarey@fl_jacarey

Miscellaneous Updates

Change 18109 on 2002/03/08 by jacarey@fl_jacarey

Updates

Change 18094 on 2002/03/08 by mdoggett@MA_MDOGGETT_LT

Update TCO section, figure and subblock descriptions. Updated TC top level figure. Added L1 Tag Index to L1 Tag.

Change 18087 on 2002/03/08 by jacarey@fl_jacarey

Updates to Pre-Fetch Parser Diagram and Pseudocode.

Change 18082 on 2002/03/08 by jacarey@fl_jacarey

Update to CP Internal Packets

Change 18077 on 2002/03/08 by jacarey@fl_jacarey

Add Input Register Stage

Change 18071 on 2002/03/08 by rvelez@rvelez_r400_win_tor

DCCIF Architectural Spec

Change 18055 on 2002/03/07 by khabbari@khabbari_r400_win

added interlace mode and supporting 2 pipe line for 20bpp

Change 18041 on 2002/03/07 by jacarey@fl_jacarey

Updates for Arbitration of Streams to Global register Bus.

Change 18032 on 2002/03/07 by jacarey@fl_jacarey

Set_State and Load_Palette are 2D State Packets

Change 18025 on 2002/03/07 by jacarey@fl_jacarey

Update to the CP_STAT register definition.

Change 17930 on 2002/03/07 by kcorrell@kcorrell_r400_docs_marlboro

update of spec

Change 17926 on 2002/03/07 by jacarey@fl_jacarey

Document that the Load_Palette packet may not be needed.

Change 17924 on 2002/03/07 by bbloemer@ma-jasonh

Revised signal naming convention to allow for MKTREE-generated prefix. Also added clock nameing convention and removed convention for instantiating flip flops.

Change 17922 on 2002/03/07 by jacarey@fl_jacarey

Updates to CP <--> MH Interface

Change 17859 on 2002/03/07 by jacarey@fl_jacarey

Update Queue Available Sizes
Update CP_STAT register definitions.

Change 17849 on 2002/03/07 by jacarey@fl_jacarey

Clarify Usage of Busy Signals Out of MIU

Change 17845 on 2002/03/07 by jasif@jasif_r400_win_tor

Changed busy signals to go to client instead of PLL model.

Change 17836 on 2002/03/07 by jasif@jasif_r400_win_tor

Changed request address to start from bit 31 instead of 29. Added separate client id's for read and write ack requests.

Change 17834 on 2002/03/07 by jasif@jasif_r400_win_tor

Added open issues: Arbitration between reads and writes, checking whether request address falls within surface information.

Change 17824 on 2002/03/07 by jacarey@fl_jacarey

Update Fetchers

Change 17822 on 2002/03/07 by jacarey@fl_jacarey

Update Buses in MIU

Change 17820 on 2002/03/07 by jacarey@fl_jacarey

Separate Path Through PFP for Real-Time

Change 17773 on 2002/03/06 by jacarey@fl_jacarey

Checkpoint Spec

Change 17771 on 2002/03/06 by nluu@nluu_r400_doclib_cnnb

- forgot to change the path

Change 17770 on 2002/03/06 by nluu@nluu_r400_doclib_cnnb

- change Last Updated field to 'file save date' instead of current date

Change 17768 on 2002/03/06 by nluu@nluu_r400_doclib_cnnb

- title => filename in header section

Change 17767 on 2002/03/06 by nluu@nluu_r400_doclib_cnnb

- change Last Updated field to 'file save date' from 'current date'

Change 17766 on 2002/03/06 by nluu@nluu_r400_doclib_cnnb

- move rom controller model test plan to test_plan subdirectory

Change 17762 on 2002/03/06 by nluu@nluu_r400_doclib_cnnb

- correct spelling and typos

Change 17760 on 2002/03/06 by nluu@nluu_r400_doclib_cnnb

- very small spelling correction

Change 17753 on 2002/03/06 by efong@efong_r400_win_tor

Added in SCL_CRTC and VGA_DISP model specs

Change 17751 on 2002/03/06 by nluu@nluu_r400_doclib_cnnb

- correct typos and stuff

Change 17736 on 2002/03/06 by jacarey@fl_jacarey

Memory Controller 0-3 each has their own go/active clock pair.

Change 17706 on 2002/03/06 by nluu@nluu_r400_doclib_cnnb

- move visio drawings to a separate directory

Change 17689 on 2002/03/06 by efong@efong_r400_win_tor

changed the pin names to i and o for input and output pins to models

Change 17670 on 2002/03/06 by jacarey@fl_jacarey

Update for Command Queue Size Update
* Registers Updated

Change 17658 on 2002/03/06 by semara@semara_r400_win_tor

doc update

Change 17632 on 2002/03/06 by gregs@gregs_r400_win_marlboro

update

Change 17629 on 2002/03/06 by rbell@rbell_crayola_win_cvd

Renamed doc

Change 17622 on 2002/03/06 by jacarey@fl_jacarey

Update RBBM FIFO depths on Top-Level Diagram

Change 17621 on 2002/03/06 by jacarey@fl_jacarey

Stall condition for wait_until.

Change 17583 on 2002/03/05 by jiezhou@jiezhou_r400_win_tor

update

Change 17582 on 2002/03/05 by bbloemer@ma-jasonh

Updated pdf file.

Change 17580 on 2002/03/05 by bbloemer@ma-jasonh

Updated the description of the ordering engine to include A/B vs C/D
fairness and the TEXTURE_WIN_COUNT.

Change 17578 on 2002/03/05 by jacarey@fl_jacarey

Checkpoint Specification

Change 17575 on 2002/03/05 by jacarey@fl_jacarey

Renamed RBBM_HI_WRTR to RBBM_HI_RDY
Added Stage_Inc for repeaters between BIF and RBBM.

Change 17556 on 2002/03/05 by rbell@rbell_crayola_win_cvd

Updates after review

Change 17549 on 2002/03/05 by jacarey@fl_jacarey

Split the Re-Ordering Queue from the Pre-fetch Parser

Change 17533 on 2002/03/05 by jacarey@fl_jacarey

Updated Streams by CP for IB2 Packet Restrictions

Change 17523 on 2002/03/05 by jacarey@fl_jacarey

Update to Fetcher Diagram

Change 17512 on 2002/03/05 by jacarey@fl_jacarey

Bitwise OR

Change 17507 on 2002/03/05 by jacarey@fl_jacarey

Update IB_PREFETCH_START packet format

Change 17504 on 2002/03/05 by scroce@scroce_r400_win_marlboro

many modifications due to changes in the next release of vcsbuild.pl

Change 17489 on 2002/03/05 by jacarey@fl_jacarey

Remove Ring State Queue

Change 17461 on 2002/03/05 by mdoggett@MA_MDOGGETT_LT

Added L1 Tag formats. Updated TCB section, figure, subblock descriptions.

Change 17452 on 2002/03/05 by semara@semara_r400_win_tor

update

Change 17450 on 2002/03/05 by bryans@bryans_crayola_doc

Update as per status meeting

Change 17449 on 2002/03/05 by semara@semara_r400_win_tor

rev 0.3

Change 17445 on 2002/03/05 by fhsien@fhsien_r400_win_marlboro

First draft of RB testbench plan

Change 17437 on 2002/03/05 by nbarbier@nbarbier_r400_win_tor

Initial Release.

Change 17436 on 2002/03/05 by jacarey@fl_jacarey

2D Scratch Memory Renamed to 2D Defaults Memory

Change 17431 on 2002/03/05 by lseiler@ma_lseiler

Version 0.7 of the Memory Format spec, with address equations, revised 2D and 3D
tiling formats, and 2D mipmap packing.

Change 17427 on 2002/03/05 by rbell@rbell_crayola_win_cvd

Print warning if linear mode is used.

Change 17424 on 2002/03/05 by jacarey@fl_jacarey

Added ROM Go/Active Pair

Change 17422 on 2002/03/05 by rvelez@rvelez_r400_win_tor

Changed address range to start from bit 31
Added Client ID to Write Ack Interface

Change 17405 on 2002/03/04 by whui@whui_r400_win_tor

updated section 10

Change 17404 on 2002/03/04 by nbarbier@nbarbier_r400_win_tor

Updated signal names.

Change 17395 on 2002/03/04 by jiezhou@jiezhou_r400_win_tor

update register list

Change 17387 on 2002/03/04 by jiezhou@jiezhou_r400_win_tor

update register list

Change 17386 on 2002/03/04 by jiezhou@jiezhou_r400_win_tor

update register list

Change 17380 on 2002/03/04 by llefebvr@llefebvre_laptop_r400

New revision of the sequencer spec.

Change 17351 on 2002/03/04 by bryans@bryans_crayola_doc

Update for 01/03/02

Change 17345 on 2002/03/04 by vliu@vliu_r400_cnvliu100_win_cvd

Update

Change 17337 on 2002/03/04 by nluu@nluu_r400_doclib_cnnb

- update

Change 17321 on 2002/03/04 by tho@tho_r400_win

updated

Change 17320 on 2002/03/04 by chwang@chwang_doc_r400_win_cvd

Update

Change 17315 on 2002/03/04 by efong@efong_r400_win_tor

Updated

Change 17313 on 2002/03/04 by jacarey@fl_jacarey

Update to CP's 2D Appendix

Change 17310 on 2002/03/04 by jasif@jasif_r400_win_tor

Updated

Change 17305 on 2002/03/04 by jacarey@fl_jacarey

Update Opcodes for Packets

Change 17190 on 2002/03/01 by lkang@lkang_r400_win_tor

updated dynamic clocking on SCLK domain.

Change 17187 on 2002/03/01 by frising@ma_frising

v.1.33
-removed support for DXV textures
-updated texture size table for new max size restriction with 3Da 128bit textures with
borders.
-closed open issue about interaction of border color with depth
and shadow textures.

Change 17184 on 2002/03/01 by jayw@MA_JAYW

weekend check-in

Change 17182 on 2002/03/01 by jacarey@fl_jacarey

Updates

Change 17169 on 2002/03/01 by mzhu@mzhu_r400_win_tor

remove support for linear surface mode

Change 17167 on 2002/03/01 by khabbari@khabbari_r400_win

small corrections to line buffer arch doc

Change 17166 on 2002/03/01 by mzhu@mzhu_r400_win_tor

remove support for linear surface mode
add more details for FIFO write control

Change 17156 on 2002/03/01 by rbell@rbell_crayola_win_cvd

Changed name of model to LB_DCP

Change 17155 on 2002/03/01 by rbell@rbell_crayola_win_cvd

Changed name of doc file.

Change 17154 on 2002/03/01 by rbell@rbell_crayola_win_cvd

update

Change 17150 on 2002/03/01 by jacarey@fl_jacarey

Update SRC_X calculation for Small_Text PM4 packet

Change 17149 on 2002/03/01 by jacarey@fl_jacarey

Updated repeater flop usage text.
STAGE_INC used instead of N_STRAP

Change 17122 on 2002/03/01 by jiezhou@jiezhou_r400_win_tor

change ICON1_X_POS to ICON1_X_START
ICON2_X_POX to ICON2_X_START
ICON1_Y_POS to ICON1_Y_START
ICON2_Y_POX to ICON2_Y_START

Change 17119 on 2002/03/01 by jowang@jowang_R400_win

added description for re-sync FIFO and some more diagrams

Change 17089 on 2002/02/28 by rbell@rbell_crayola_win_cvd

Removed sub-word write enable

Change 17084 on 2002/02/28 by jacarey@fl_jacarey

Fix Typo in the IM_Load and IM_Load_Immediate packets

Change 17066 on 2002/02/28 by jacarey@fl_jacarey

1. Added IM_LOAD_IMMEDIATE packet.

2. Update to HOSTDATA_BLT2 and HOSTDATA_BLT_PNTR packets.

Change 17053 on 2002/02/28 by jiezhou@jiezhou_r400_win_tor

update

Change 17016 on 2002/02/28 by jacarey@fl_jacarey

Update Interrupt Signals Into RBBM

Change 17010 on 2002/02/28 by rbell@rbell_crayola_win_cvd

Initial draft (and merge from older docs).

Change 16932 on 2002/02/27 by jowang@jowang_R400_win

Block Spec: read buffer, re-sync FIFO, p2s sync, and line controller
diagram + description for read buffer

Change 16927 on 2002/02/27 by frising@ma_frising

Crap! fix cut and paste error. No version change.

Change 16925 on 2002/02/27 by frising@ma_frising

v.1.32
-removed TT_CMD from texture instruction.

Change 16924 on 2002/02/27 by lkang@lkang_r400_win_tor

Added power state managment. section 6.0

Change 16917 on 2002/02/27 by lkang@lkang_r400_win_tor

Added 5.2 for dynamic clocking in display clocks

Change 16914 on 2002/02/27 by jiezhou@jiezhou_r400_win_tor

update after spec review Feb. 26

Change 16906 on 2002/02/27 by jacarey@fl_jacarey

Version 0.02 has Pre-Fetch Parser Algorithm Updates

Change 16897 on 2002/02/27 by tho@tho_r400_win

updated with section regarding the IO Model generating Perl Script

Change 16896 on 2002/02/27 by jacarey@fl_jacarey

Updates to Packets and Pseudocode for Pre-Fetch Parser

Change 16878 on 2002/02/27 by mpersaud@mpersaud_r400_win_tor

* Updated link to Figure 1.2-1 VIP DMA Top Level Diagram

Change 16867 on 2002/02/27 by mpersaud@mpersaud_r400_win_tor

* Updated Section 4.1 Pin interface, added DCIO enable signals
* Updated Appendix for delta between R300 and R400

Change 16859 on 2002/02/27 by lseiler@ma_lseiler

new 3D tile patterns and initial 3D equations

Change 16856 on 2002/02/27 by jacarey@fl_jacarey

Updates

Change 16850 on 2002/02/27 by jacarey@fl_jacarey

Pre-Fetch Parser Top-Level State Machine

Change 16849 on 2002/02/27 by khabbari@khabbari_r400_win

correction to read pointer section in line buffer arch doc

Change 16843 on 2002/02/27 by rbell@rbell_crayola_win_cvd

Updates to macro name and pin out

Change 16815 on 2002/02/27 by markf@markf_r400_win_marlboro

Updated Analog section to R300 numbers. Fixed quantity of DLL's for RV400 and R400. Removed LVDS from R400.

Change 16809 on 2002/02/27 by nluu@nluu_r400_doclib_cnnb

- update

Change 16799 on 2002/02/27 by nluu@nluu_r400_doclib_cnnb

- update

Change 16794 on 2002/02/27 by jacarey@fl_jacarey

Updates to 2D Packets

Change 16793 on 2002/02/27 by mmantor@mmantor_r400_win

updated memory areas based on the memory compilation of memories in the following blocks for the VGT, PA, SC blocks and updated labeling for current planned blocks

Change 16779 on 2002/02/26 by semara@semara_r400_win_tor

update doc

Change 16778 on 2002/02/26 by semara@semara_r400_win_tor

doc updates

Change 16762 on 2002/02/26 by jayw@MA_JAYW

typo fixes

Change 16734 on 2002/02/26 by jacarey@fl_jacarey

Checkpoint Specifications

Change 16731 on 2002/02/26 by lseiler@ma_lseiler

Spreadsheets of 2D address conversions

Change 16730 on 2002/02/26 by lseiler@ma_lseiler

Updated 1D and 2D formats and address equations

Change 16718 on 2002/02/26 by efong@efong_r400_win_tor

updated io spec due to change in dcio spec

Change 16710 on 2002/02/26 by jacarey@fl_jacarey

Updates to packets.

Change 16688 on 2002/02/26 by jacarey@fl_jacarey

Update Diagram

Change 16661 on 2002/02/26 by jacarey@fl_jacarey

Update RTEE Diagram (Gated and Permanent Clocks)

Change 16660 on 2002/02/26 by rbell@rbell_crayola_win_cvd

First draft of what env will look like.

Change 16658 on 2002/02/26 by mpersaud@mpersaud_r400_win_tor

* added Section 4.1 Pin interface
* updated Appendix for delta between R300 and R400
* update Figure 1.2-1 VIP Top Level Diagram to move uvip_rbbmif to DC level and rename to RBBMIF

Change 16657 on 2002/02/26 by mpersaud@mpersaud_r400_win_tor

Move uvip_rbbmif to DC level and rename to RBBMIF

Change 16653 on 2002/02/26 by jacarey@fl_jacarey

Update Interfaces to RBBM from BIF and CP
Fix Type on MIU Diagram

Change 16648 on 2002/02/26 by jhoule@MA_JHOULE

v1.31:
Brought back DIM_3D in case we get rid of dual format 3D texture (if noise gets dropped out).
Moved BORDER_* together.
Only affects TFetch Const.

Change 16646 on 2002/02/26 by jiezhou@jiezhou_r400_win_tor

update

Change 16632 on 2002/02/26 by nbarbier@nbarbier_r400_win_tor

Updated document after resolving open issues.

Change 16602 on 2002/02/26 by jacarey@fl_jacarey

Updated clock name for RBBM

Change 16565 on 2002/02/25 by jiezhou@jiezhou_r400_win_tor

update

Change 16552 on 2002/02/25 by dclifton@dclifton_r400

Updates to interface, drawings, lines, points, and polymode.

Change 16546 on 2002/02/25 by khabbari@khabbari_r400_win

added more details to the lb arch doc

Change 16537 on 2002/02/25 by rbell@rbell_crayola_win_cvd

Updates after review

Change 16533 on 2002/02/25 by markf@markf_r400_win_marlboro

Updated SX w/ parameter caches

Change 16521 on 2002/02/25 by rbell@rbell_crayola_win_cvd

update

Change 16505 on 2002/02/25 by jacarey@fl_jacarey

Checkpoint CP Spec.

Change 16503 on 2002/02/25 by jacarey@fl_jacarey

Update Command Stream Fetcher Diagram

Change 16499 on 2002/02/25 by jimmylau@jimmylau_r400_win_tor

Add a section on stereoscopic display. Change CRTC naming convention to CRTC0 / CRTC1.

Change 16495 on 2002/02/25 by jacarey@fl_jacarey

Update Queues on Diagram for PFP

Change 16494 on 2002/02/25 by mpersaud@mpersaud_r400_win_tor

added state machine appendix

Change 16493 on 2002/02/25 by mpersaud@mpersaud_r400_win_tor

update to add uvip_rbbmif

Change 16492 on 2002/02/25 by mpersaud@mpersaud_r400_win_tor

general clean up

Change 16490 on 2002/02/25 by mpersaud@mpersaud_r400_win_tor

* added Appendix - R300 vs. R400 delta
* updated VIP Top Level diagram
* updated structure section

Change 16489 on 2002/02/25 by mpersaud@mpersaud_r400_win_tor

* moved uvipdmaregs into uvipdma

* renamed uvip_rbbm to uvip_rbbmif
* added ORing of interrupts

Change 16488 on 2002/02/25 by vliu@vliu_r400_cnvliu100_win_cvd

Update

Change 16484 on 2002/02/25 by rbell@rbell_crayola_win_cvd

updated

Change 16482 on 2002/02/25 by chwang@chwang_r400_win_cvd

Update.

Change 16470 on 2002/02/25 by tho@tho_r400_win

update

Change 16469 on 2002/02/25 by jasif@jasif_r400_win_tor

Updated

Change 16462 on 2002/02/25 by efong@efong_r400_win_tor

Updated

Change 16450 on 2002/02/25 by jacarey@fl_jacarey

Add ROM pins back to the RBBM Spec.

Change 16412 on 2002/02/22 by nluu@nluu_r400_doclib_cnnb

- update

Change 16411 on 2002/02/22 by jiezhou@jiezhou_r400_win_tor

1) add Overlay Format Expansion block
2) remove dynamic expansion for Graphic/overlay keyer
3) add host read/write description for LUT palette
4) add ignore_alpha for graphic/overlay keyer
5) update graphic/overlay Mux function
6) add constant format for matrix conversion

Change 16407 on 2002/02/22 by lkang@lkang_r400_win_tor

Added dynamic clocking on display clock domain. sec. 5.1

Change 16359 on 2002/02/22 by gregs@gregs_r400_win_marlboro

Added interface to CG_PM.

Change 16349 on 2002/02/22 by gregs@gregs_r400_win_marlboro

daily update.

Change 16322 on 2002/02/22 by lseiler@ma_lseiler

Updated with 1D and 2D access equations

Change 16319 on 2002/02/22 by jacarey@fl_jacarey

Removed ROM's RTR signals from RBBM interface. BIF forwards transactions to the MH instead of through the RBBM.

Change 16309 on 2002/02/22 by jacarey@fl_jacarey

Update RBIU to show that read return data is registed on permanent clock.

Change 16304 on 2002/02/22 by jacarey@fl_jacarey

1. Updated interrupt signals to the RBBM

** Senders of interrupts need to do their own combining.

Change 16288 on 2002/02/22 by bryans@bryans_crayola_doc

Update VideoIP DV Methodology to include CRTC configuration

Change 16284 on 2002/02/22 by efong@efong_r400_win_tor

added in sentence for MV_DISABLE and VGA_DISABLE to be hooked up to BIF model

Change 16283 on 2002/02/22 by efong@efong_r400_win_tor

fixed up the spec due to the 0.1 version of the power management spec

Change 16217 on 2002/02/21 by fhsien@fhsien_r400_win_marlboro

Added command chart
change runtest --> runtest.pl

Change 16213 on 2002/02/21 by rbell@rbell_crayola_win_cvd

First draft of this

Change 16210 on 2002/02/21 by mdoggett@MA_MDOGGETT_LT

Updated formats. Added 2d interlace format and 3d four layer format. Added L2 read synchronisation description.

Change 16207 on 2002/02/21 by frising@ma_frising

v.1.30

Change 16194 on 2002/02/21 by lkang@lkang_r400_win_tor

Power management spec for DC

Change 16178 on 2002/02/21 by hartogs@fl_hartogs

Minor Cleanup

Change 16177 on 2002/02/21 by jacarey@fl_jacarey

No Change

Change 16175 on 2002/02/21 by rbell@rbell_crayola_win_cvd

More updates since the last reviews

Change 16174 on 2002/02/21 by jacarey@fl_jacarey

Checkpoint All Specifications

Change 16170 on 2002/02/21 by hartogs@fl_hartogs

Fixed up some tessellation stuff.

Change 16147 on 2002/02/21 by scroce@scroce_r400_win_marlboro

Some small changes to the doc

Change 16146 on 2002/02/21 by rbell@rbell_crayola_win_cvd

Updates to both palette models

Change 16145 on 2002/02/21 by mpersaud@mpersaud_r400_win_tor

Initial Revision

Change 16142 on 2002/02/21 by mpersaud@mpersaud_r400_win_tor

Changed Text Box

Change 16138 on 2002/02/21 by jacarey@fl_jacarey

Revision of RBBM and CP areas

Change 16109 on 2002/02/20 by dwong@cndwong2

version 0.4

Change 16106 on 2002/02/20 by nluu@nluu_r400_doclib_cnnb

- update

Change 16100 on 2002/02/20 by nluu@nluu_r400_doclib_cnnb

- update

Change 16095 on 2002/02/20 by fhsien@fhsien_r400_win_marlboro

DOC for Runtest Script

Change 16083 on 2002/02/20 by jacarey@fl_jacarey

Updated CP and RBBM Areas

Change 16079 on 2002/02/20 by gregs@gregs_r400_win_marlboro

daily update.

Change 16075 on 2002/02/20 by mpersaud@mpersaud_r400_win_tor

General update and/or addition to depot.

Change 16053 on 2002/02/20 by rbell@rbell_crayola_win_cvd

New spec for R400

Change 16041 on 2002/02/20 by scroce@scroce_r400_win_marlboro

Clearing up some things once again

Change 16033 on 2002/02/20 by csampayo@fl_csampayo_r400

Updated VGT and SU sections and schedule

Change 16009 on 2002/02/20 by scroce@scroce_r400_win_marlboro

Trying to clarify the setup doc

Change 16007 on 2002/02/20 by rramsey@RRAMSEY_P4_r400_win

Update rect description

Change 15992 on 2002/02/20 by efong@efong_r400_win_tor

changed from dispout inject to scaler inject model

Change 15972 on 2002/02/20 by jasif@jasif_r400_win_tor

Partially updated after document review.

Change 15952 on 2002/02/19 by ctaylor@fl_ctaylor_r400_win_marlboro

Most of top-level arch done.

Change 15923 on 2002/02/19 by bryans@bryans_crayola_win_cvd

Add test plan templates for VID core

Change 15920 on 2002/02/19 by bryans@bryans_crayola_win_cvd

Update for this week

Change 15919 on 2002/02/19 by efong@efong_r400_win_tor

fixed up the spec due to the review

Change 15914 on 2002/02/19 by gregs@gregs_r400_win_marlboro

Update.

Change 15878 on 2002/02/19 by bryans@bryans_crayola_win_cvd

Source tree for display core

Change 15862 on 2002/02/19 by efong@efong_r400_win_tor

renamed top level diagram from hdcp disable to hdcp_enable
changed the forciable signals section to be more standardized

Change 15860 on 2002/02/19 by ygiang@ygiang_r400_win_marlboro

added: Ferrt Shader Docs and Test environment cook book

Change 15851 on 2002/02/19 by efong@efong_r400_win_tor

updated due to review and some pad changes

Change 15837 on 2002/02/19 by tho@tho_r400_win

updated

Change 15828 on 2002/02/19 by nbarbier@nbarbier_r400_win_tor

Initial revision of DCIO Interface.

Change 15770 on 2002/02/18 by beiwang@bei_mcspec

deleted CAM and address return queues from mcci diagram, modified the cam diagram, added note on bank ABCD->bank 0,1,2,3

Change 15752 on 2002/02/18 by tho@tho_r400_win

update cursor test plan

Change 15742 on 2002/02/18 by jasif@jasif_r400_win_tor

Updated.

Change 15741 on 2002/02/18 by bryans@bryans_crayola_win_cvd

Update status/dates

Change 15727 on 2002/02/18 by vliu@vliu_r400_cnvliu100_win_cvd

Update the status.

Change 15723 on 2002/02/18 by efong@efong_r400_win_tor

Updated dates

Change 15720 on 2002/02/18 by chwang@chwang_r400_win_cvd

Update.

Change 15718 on 2002/02/18 by efong@efong_r400_win_tor

after final review ...

Change 15716 on 2002/02/18 by jasif@jasif_r400_win_tor

Added new forcible signals table.

Change 15713 on 2002/02/18 by jasif@jasif_r400_win_tor

Updated Error section. Added open issue on dynamic clocking.

Change 15711 on 2002/02/18 by tho@tho_r400_win

update

Change 15709 on 2002/02/18 by rbell@rbell_crayola_win_cvd

Changes

Change 15689 on 2002/02/16 by jimmylau@jimmylau_r400_win_tor

Major revision of CRTC block specs after the first specs review.

Change 15679 on 2002/02/15 by jiezhou@jiezhou_r400_win_tor

modified state machine

Change 15678 on 2002/02/15 by jiezhou@jiezhou_r400_win_tor

add 32 bpp digital output for graphic request

Change 15676 on 2002/02/15 by jiezhou@jiezhou_r400_win_tor

Initial release

Change 15671 on 2002/02/15 by jacarey@fl_jacarey

Checkpoint Specifications

Change 15669 on 2002/02/15 by mzhu@mzhu_r400_win_tor

initial version of DMIF block spec

Change 15667 on 2002/02/15 by mzhu@mzhu_r400_win_tor

define pixel order for tiling mode with one DCP pipeline enabled

Change 15664 on 2002/02/15 by kcorrell@kcorrell_r400_docs_marlboro

Updated GART area in MH

Change 15658 on 2002/02/15 by gregs@gregs_r400_win_marlboro

Added parallel ROM support on the DVO interface.

Change 15627 on 2002/02/15 by gregs@gregs_r400_win_marlboro

ROM+DEBUG blocks: increased are by 25% and added 640x8 SRAM area.

Change 15593 on 2002/02/15 by lseiler@ma_lseiler

Restored the unrevised text that I deleted for review version 0.4, removed change bars

Change 15586 on 2002/02/15 by gregs@gregs_r400_win_marlboro

increased CG area by 50% (an estimate for new power management functionality).

Change 15585 on 2002/02/15 by jacarey@fl_jacarey

Clarify Byte Enable Support Through the RBBM.

Change 15570 on 2002/02/15 by jacarey@fl_jacarey

1. Skew Control Update
2. Update Command FIFO Depth

Change 15564 on 2002/02/15 by kcorrell@kcorrell_r400_docs_marlboro

updated MH area

Change 15563 on 2002/02/15 by efong@efong_r400_win_tor

closed some open issues

Change 15562 on 2002/02/15 by efong@efong_r400_win_tor

updated a ton of things

Change 15544 on 2002/02/14 by rvelez@rvelez_r400_win_tor

Added more details to the spec

Change 15543 on 2002/02/14 by jasif@jasif_r400_win_tor

Updated.

Change 15541 on 2002/02/14 by gregs@gregs_r400_win_marlboro

daily changes

Change 15534 on 2002/02/14 by csampayo@fl_csampayo_r400

Updated SU section, both docs

Change 15533 on 2002/02/14 by beiwang@bei_r400_area

Added some MC skid buffer

Change 15529 on 2002/02/14 by beiwang@bei_r400_area

Added MC changes

Change 15508 on 2002/02/14 by vromaker@MA_VIC_P4

updated SQ macro area numbers to those obtained with .13 compiler

Change 15492 on 2002/02/14 by jasif@jasif_r400_win_tor

Updated with recommendations made during the review.

Change 15455 on 2002/02/14 by rbell@rbell_crayola_win_cvd

Added open issues, and changes as per Jie.

Change 15450 on 2002/02/14 by rbell@rbell_crayola_win_cvd

Changes after the DMIF design review

Change 15446 on 2002/02/14 by jacarey@fl_jacarey

Update Diagrams

Change 15418 on 2002/02/13 by csampayo@fl_csampayo_r400

Updated most of the SU section.

Change 15404 on 2002/02/13 by jacarey@fl_jacarey

Checkpoint CP Specifications

Change 15392 on 2002/02/13 by jacarey@fl_jacarey

Update Interface to Memory Hub on Diagram

Change 15390 on 2002/02/13 by gregs@gregs_r400_win_marlboro

daily update

Change 15387 on 2002/02/13 by gregs@gregs_r400_win_marlboro

daily update

Change 15377 on 2002/02/13 by jacarey@fl_jacarey

Update Internal Register Bus Connections

Change 15375 on 2002/02/13 by jasif@jasif_r400_win_tor

Updated.

Change 15371 on 2002/02/13 by jacarey@fl_jacarey

Minor Updates to RBBM.

Change 15369 on 2002/02/13 by jacarey@fl_jacarey

Update Diagrams

Change 15358 on 2002/02/13 by gregs@gregs_r400_win_marlboro

daily update

Change 15333 on 2002/02/13 by jacarey@fl_jacarey

Updates to RTEE Diagram

Change 15325 on 2002/02/13 by jacarey@fl_jacarey

Update for Review

Change 15304 on 2002/02/12 by tho@tho_r400_win

removed MV_DISABLE strap from doc.

Change 15301 on 2002/02/12 by tho@tho_r400_win

Changed strap value generation, updated forcible signal section

Change 15297 on 2002/02/12 by jacarey@fl_jacarey

Incremental Update to the Spec.

Change 15296 on 2002/02/12 by jacarey@fl_jacarey

Revision 0.06 of RBBM Spec

Change 15295 on 2002/02/12 by hartogs@fl_hartogs

Updated the RBBM interface table.

Change 15281 on 2002/02/12 by jacarey@fl_jacarey

Update Diagrams

Change 15279 on 2002/02/12 by jasif@jasif_r400_win_tor

Updated dynamic clocking document. Changed file name from RBBM_model.doc to RBBMIF_model.doc.

Change 15276 on 2002/02/12 by jacarey@fl_jacarey

Update connections to command stream fetcher

Change 15273 on 2002/02/12 by jacarey@fl_jacarey

Separate RB_RPTR write requests to MIU

Change 15271 on 2002/02/12 by jacarey@fl_jacarey

Updates to Diagram of RBBM Top-Level

Change 15265 on 2002/02/12 by gregs@gregs_r400_win_marlboro

new spec - initial version.

Change 15264 on 2002/02/12 by efong@efong_r400_win_tor

Fixed up IO model after new review

Change 15222 on 2002/02/12 by jhoule@MA_JHOULE

0.9.5:
Added WR_MASK_{XYZW} in SQ<--TPC interface.
Updated TOC and LOF.

Change 15216 on 2002/02/12 by efong@efong_r400_win_tor

Updated PLL model because of the review as well as added in GO and BUSY signals

Change 15215 on 2002/02/12 by efong@efong_r400_win_tor

put documentation for VIP_DEVICE being inverted.

Change 15191 on 2002/02/11 by jasif@jasif_r400_win_tor

Changed name of document.

Change 15190 on 2002/02/11 by jasif@jasif_r400_win_tor

Updated.

Change 15188 on 2002/02/11 by bryans@bryans_crayola_win_cvd

Update with feedback from meeting; add model code reviews, weekly issues

Change 15182 on 2002/02/11 by nluu@nluu_r400_cnnb

- update

Change 15167 on 2002/02/11 by jasif@jasif_r400_win_tor

Added section on avoiding race conditions and behavioural coding styles for performance.

Change 15166 on 2002/02/11 by jacarey@fl_jacarey

Update Waveforms for Slow Clients

Change 15165 on 2002/02/11 by gregs@gregs_r400_win_marlboro

Corrected typos.

Change 15164 on 2002/02/11 by khabbari@khabbari_r400_win

the first released version of crtc tv interface document.
Also added some details to line buffer arch document.

Change 15156 on 2002/02/11 by scroce@scroce_r400_win_marlboro

Added some little things about v2html

Change 15154 on 2002/02/11 by jacarey@fl_jacarey

Remove Skew Control Signal Between RBBM and CP

Change 15151 on 2002/02/11 by dclifton@dclifton_r400

Updates to drawings

Change 15139 on 2002/02/11 by nluu@nluu_r400_cnnb

- update

Change 15133 on 2002/02/11 by nluu@nluu_r400_cnnb

- update

Change 15126 on 2002/02/11 by nluu@nluu_r400_cnnb

- update

Change 15121 on 2002/02/11 by csampayo@fl_csampayo_lt_r400

SU updates

Change 15120 on 2002/02/11 by nluu@nluu_r400_cnnb

- update status

Change 15115 on 2002/02/11 by hartogs@fl_hartogs

Refreshed early diagrams and text.

Change 15108 on 2002/02/11 by bryans@bryans_crayola_win_cvd

Update milestone dates

Change 15106 on 2002/02/11 by nluu@nluu_r400_cnnb

- check in so I don't lose the changes

Change 15100 on 2002/02/11 by jasif@jasif_r400_win_tor

Updated.

Change 15099 on 2002/02/11 by efong@efong_r400_win_tor

Updated DV status

Change 15096 on 2002/02/11 by dglen@dglen_r400_dell

Added section on update and control of double buffered register fields.

Change 15095 on 2002/02/11 by vliu@vliu_r400_cnvliu100_win_cvd

Updating the dates.

Change 15091 on 2002/02/11 by dclifton@dclifton_r400

Changed a few diagrams and equations slightly

Change 15079 on 2002/02/11 by chwang@chwang_r400_win_cvd

Update

Change 15073 on 2002/02/11 by tho@tho_r400_win

updated status report

Change 15068 on 2002/02/11 by dclifton@dclifton_r400

Updated top level block diagram

Change 15067 on 2002/02/11 by rbell@rbell_crayola_win_cvd

Updates

Change 15066 on 2002/02/11 by wlawless@wlawless

Nothing major

Change 15018 on 2002/02/08 by jasif@jasif_r400_win_tor

Initial revision.

Change 15017 on 2002/02/08 by rvelez@rvelez_r400_win_tor

Added more description for the interface

Change 15016 on 2002/02/08 by gregs@gregs_r400_win_marlboro

Updated after David Glen and John Carey reviews.

Change 15002 on 2002/02/08 by jowang@jowang_R400_win

added a column for # of horizontal filter taps

Change 15001 on 2002/02/08 by jowang@jowang_R400_win

updated with overscan and interlace interface signals

Change 15000 on 2002/02/08 by jasif@jasif_r400_win_tor

Updated Joveria's status.

Change 14967 on 2002/02/08 by khabbari@khabbari_r400_win

added write/read pointer des to the LB ARCH doc

Change 14962 on 2002/02/08 by gregs@gregs_r400_win_marlboro

ROM spec is ready for verilog coding phase.

Change 14961 on 2002/02/08 by efong@efong_r400_win_tor

Updated document after initial review

Change 14959 on 2002/02/08 by rbell@rbell_crayola_win_cvd

Added sample test code.

Change 14955 on 2002/02/08 by rbell@rbell_crayola_win_cvd

Initial draft

Change 14948 on 2002/02/08 by rbell@rbell_crayola_win_cvd

Updates after review

Change 14934 on 2002/02/08 by paulv@MA_PVELLA

Updated MHS with most recent changes and updates.

Change 14912 on 2002/02/07 by csampayo@fl_csampayo_r400

Added unit/function owners and hyperlinks to the R400_PA_Functional_Validation_Approach_Plan document

Change 14904 on 2002/02/07 by jasif@jasif_r400_win_tor

Modified dynamic clock section.

Change 14895 on 2002/02/07 by hartogs@fl_hartogs

Completed description of de-stripping, de-fanning, decomposition of quads and polygons into triangles, prprovoking vertex (flat shading), and line stipple wireframe fill mode issues.

Change 14883 on 2002/02/07 by jhoule@MA_JHOULE

Added 32-bit channels blending.
Described math foundations.
Updated TOC.

Change 14878 on 2002/02/07 by grayc@grayc_r400_win

initial checkin

Change 14877 on 2002/02/07 by grayc@grayc_r400_win

moving files

Change 14861 on 2002/02/07 by grayc@grayc_r400_win

Initial checkin

Change 14855 on 2002/02/07 by mdoggett@MA_MDOGGETT_LT

Updates according to TC spec review and many more discussion. Changes to TCD figure, L1 and L2 tags, L2 formats, padding and degamma.

Change 14849 on 2002/02/07 by efong@efong_r400_win_tor

Made some changes due to the review of the strap model.

Change 14842 on 2002/02/07 by rbell@rbell_crayola_win_cvd

Changed CP to DCP

Change 14824 on 2002/02/07 by jhoule@MA_JHOULE

Forgot the usage tables. Now updated!

Change 14823 on 2002/02/07 by jhoule@MA_JHOULE

Moved fields (VTX_FORMAT, EXP_ADJUST) from VFetch instr so that they are byte-aligned.

Change 14818 on 2002/02/07 by jhoule@MA_JHOULE

Aesthetic

Change 14817 on 2002/02/07 by jhoule@MA_JHOULE

Brought SRC_SEL* in same DWORD.

Change 14806 on 2002/02/06 by jiezhou@jiezhou_r400_win_tor

update window control

Change 14797 on 2002/02/06 by efong@efong_r400_win_tor

New version of the IO model with the names fixed up and some of the issues closed.

Change 14796 on 2002/02/06 by jhoule@MA_JHOULE

Changed instruction fields so that they match between VFetch and TFetch.
This helps the SQ (fields are always at the same place).
Also added cool macro for bit numbers.

Change 14785 on 2002/02/06 by bbloemer@ma-jasonh

Removed address return bus and base and coord from RB access bus.

Change 14698 on 2002/02/05 by jacarey@fl_jacarey

Checkpoint the CP's PM4 Specification

Change 14697 on 2002/02/05 by jimmylau@jimmylau_r400_win_tor

Rev 0.3. Decided to keep notation of internal & external overscan.

Change 14696 on 2002/02/05 by fhsien@fhsien_r400_win_marlboro

Add Running eum_only. (no build is done).

Change 14694 on 2002/02/05 by jacarey@fl_jacarey

Checkpoint 2D Appendix

Change 14691 on 2002/02/05 by jacarey@fl_jacarey

Checkpoint RBBM Spec

Change 14687 on 2002/02/05 by jacarey@fl_jacarey

Update CP's Memory Interface Unit Diagram

Change 14685 on 2002/02/05 by jacarey@fl_jacarey

Update RBBM Top-Level Diagram

1. Slip Buffer Registers
2. Add RBBM_CP_DMA_DRAW_DEC signal
3. Update Switch to Send Host Data through Non-Queued Path

Change 14677 on 2002/02/05 by hartogs@fl_hartogs

Added more information on the primitive types and provoking vertex.

Change 14674 on 2002/02/05 by rbell@rbell_crayola_win_cvd

initial draft

Change 14667 on 2002/02/05 by jacarey@fl_jacarey

Update

Change 14661 on 2002/02/05 by jacarey@fl_jacarey

Update PFP to ME Diagram
Remove Diagram for Old Fetchers

Change 14643 on 2002/02/05 by khabbari@khabbari_r400_win

added a block diagram for write data path in the line buffer doc

Change 14631 on 2002/02/05 by tho@tho_r400_win

updated Data Processing Algorithms section regarding PLI/CLI usage

Change 14621 on 2002/02/05 by efong@efong_r400_win_tor

Updated PLL model to support stop and go signals

Change 14587 on 2002/02/04 by khabbari@khabbari_r400_win

added some description to data processing algorithm in line buffer arch document

Change 14582 on 2002/02/04 by jasif@jasif_r400_win_tor

Updated finish dates.

Change 14568 on 2002/02/04 by vromaker@MA_VIC_P4

updates

Change 14551 on 2002/02/04 by jasif@jasif_r400_win_tor

Updated.

Change 14543 on 2002/02/04 by rbell@rbell_crayola_win_cvd

Final draft

Change 14540 on 2002/02/04 by gregs@gregs_r400_win_marlboro

Added/updated after implementation analysis and adding parallel PROM support.

Change 14528 on 2002/02/04 by frising@ma_frising

v.1.19
-Fixed up how certain texture formats go into the L2 cache in order
to guarantee consistency within a texture family (e.g. 8, 8_8,
8_8_8_8).

Change 14527 on 2002/02/04 by jasif@jasif_r400_win_tor

Updated.

Change 14526 on 2002/02/04 by jasif@jasif_r400_win_tor

Added section on dynamic clocking.

Change 14509 on 2002/02/04 by paulv@MA_PVELLA

Updated block diagram to match changes and updates to the MHS section in the spec.

Change 14500 on 2002/02/04 by nluu@nluu_r400_cnnb

- update

Change 14499 on 2002/02/04 by rbell@rbell_crayola_win_cvd

Added assumptions, stubs, model terminology, etc.

Change 14495 on 2002/02/04 by jacarey@fl_jacarey

Updates to RTEE Event Engine Diagram

Change 14492 on 2002/02/04 by vromaker@MA_VIC_P4

updates, new figures

Change 14481 on 2002/02/04 by chwang@chwang_r400_win_cvd

Update.

Change 14476 on 2002/02/04 by rbell@rbell_crayola_win_cvd

Updates

Change 14475 on 2002/02/04 by rbell@rbell_crayola_win_cvd

Updates after reviews, added errors warnings and assumptions.

Change 14474 on 2002/02/04 by jimmylau@jimmylau_r400_win_tor

Revision 0.2 of CRTC block specs. Remove notation of external/internal overscan. Overscan is always outside the active display area.

Change 14472 on 2002/02/04 by jasif@jasif_r400_win_tor

Updated.

Change 14470 on 2002/02/04 by tho@tho_r400_win

updated status report

Change 14469 on 2002/02/04 by vromaker@MA_VIC_P4

new figures

Change 14468 on 2002/02/04 by efong@efong_r400_win_tor

Updated

Change 14466 on 2002/02/04 by llefebvr@llefebvre_laptop_r400

Version 1.7 of the Sequencer spec.

Change 14465 on 2002/02/03 by jiezhou@jiezhou_r400_win_tor

Add some control management

Change 14464 on 2002/02/03 by vromaker@MA_VIC_P4

changes

Change 14444 on 2002/02/01 by jasif@jasif_r400_win_tor

Updated open issues.

Change 14441 on 2002/02/01 by jasif@jasif_r400_win_tor

Updated after spec review.

Change 14440 on 2002/02/01 by efong@efong_r400_win_tor

Added in IO model and strap model

Change 14439 on 2002/02/01 by jasif@jasif_r400_win_tor

Updated

Change 14436 on 2002/02/01 by mzhu@mzhu_r400_win_tor

change CP to DCP

Change 14426 on 2002/02/01 by rbagley@MA_RBAGLEY_LTXP

0.1 (d) Intermediate check-in. Additional comments. The bloat of the container doc has been remedied.

Change 14424 on 2002/02/01 by jacarey@fl_jacarey

Checkpoint Specifications

Change 14416 on 2002/02/01 by khabbari@khabbari_r400_win

small fixes

Change 14411 on 2002/02/01 by khabbari@khabbari_r400_win

The first draft of line buffer doc

Change 14402 on 2002/02/01 by rbell@rbell_crayola_win_cvd

Changes some arch/conf names, added BFMs section.

Change 14400 on 2002/02/01 by jasif@jasif_r400_win_tor

Updated

Change 14399 on 2002/02/01 by semara@semara_r400_win_tor

rev 2

Change 14395 on 2002/02/01 by rbagley@MA_RBAGLEY_LTXP

1.0 (c) Revisions to instructions and miscellaneous corrections.

Change 14379 on 2002/02/01 by nbarbier@nbarbier_r400_win_tor

First revision of CRTC/DISPOUT interface

Change 14377 on 2002/02/01 by frising@ma_frising

-Change GetBorderColorPercentage to GetBorderColorFraction. Thanks
Jocelyn.
-no version change.

Change 14366 on 2002/01/31 by jimmylau@jimmylau_r400_win_tor

Initial revision

Change 14365 on 2002/01/31 by frising@ma_frising

v.1.18
-Added 6 new multimedia texture formats
-remove FetchWhiteTexture, replace with GetBorderColorPercentage
-remove mask option from DST_SEL_ in texture constant
-move mask option of DST_SEL in insts from 6 to 7
-Add new YCbCr black border color

Change 14363 on 2002/01/31 by vromaker@MA_VIC_P4

new

Change 14362 on 2002/01/31 by vromaker@MA_VIC_P4

updates

Change 14356 on 2002/01/31 by hartogs@fl_hartogs

Still in progress.
New block diagrams.
Checked-in for backup.

Change 14339 on 2002/01/31 by jacarey@fl_jacarey

Update Busy Signals on Diagrams

Change 14333 on 2002/01/31 by chwang@chwang_r400_win_cvd

Updated Status.

Change 14313 on 2002/01/31 by efong@efong_r400_win_tor

Fixed up a ton of things from the review meeting

Change 14301 on 2002/01/31 by lkang@lkang_r400_win_tor

new directories

Change 14300 on 2002/01/31 by lkang@lkang_r400_win_tor

This directory is for DC-MH interface block architecture and implementation specs

Change 14299 on 2002/01/31 by wlawless@wlawless

This version was updated to match the new Arch spec. The diagrams were updated.. The depth logic is still old because we are working the color right now.

Change 14292 on 2002/01/30 by rbagley@MA_RBAGLEY_LTXP

1.0 (b) Further revision.

Change 14287 on 2002/01/30 by dwong@cndwong2

xDCT document, v0.2

Adding in implementation details as well as updating open issues.

Change 14275 on 2002/01/30 by rbell@rbell_crayola_win_cvd

more changes

Change 14267 on 2002/01/30 by lseiler@ma_lseiler

Version 0.4: major update of sections 1-3, remaining sections temporarily removed until they can be fully updated.

Change 14265 on 2002/01/30 by jacarey@fl_jacarey

Streams Processed by CP Diagram

Change 14263 on 2002/01/30 by jacarey@fl_jacarey

Baseline New Documents in Perforce

Change 14260 on 2002/01/30 by jacarey@fl_jacarey

Checkpoint Specifications

Change 14250 on 2002/01/30 by nluu@nluu_r400_cnnb

- fix typo

Change 14244 on 2002/01/30 by gregs@gregs_r400_win_marlboro

Updated with the new CP based power management ideas. Added startup description.

Change 14239 on 2002/01/30 by khabbari@khabbari_r400_win

added some description in "Interface Functional Description and Purpose" section

Change 14238 on 2002/01/30 by jasif@jasif_r400_win_tor

Updated.

Change 14237 on 2002/01/30 by askende@andi_r400_docs

new rev of the document.

Change 14236 on 2002/01/30 by tho@tho_r400_win

updated rom controller model test plan

Change 14212 on 2002/01/30 by nluu@nluu_r400_cnnb

- update

Change 14202 on 2002/01/29 by dclifton@dclifton_r400

Updated with new diagrams to match pinout, added zoffset info

Change 14198 on 2002/01/29 by jiezhou@jiezhou_r400_win_tor

remove

Change 14197 on 2002/01/29 by jiezhou@jiezhou_r400_win_tor

to remove CP

Change 14196 on 2002/01/29 by jiezhou@jiezhou_r400_win_tor

Change name CP to DCP

Change 14193 on 2002/01/29 by jasif@jasif_r400_win_tor

Initial Revision of DCCIF Model Spec

Change 14184 on 2002/01/29 by mdoggett@MA_MDOGGETT_LT

Updated TCD section. New top level diagram. Algorithm for DXT added.

Change 14183 on 2002/01/29 by rthambim@rthambim_r400_win_tor

Modified naming convention and removed SWAP bits.

Change 14180 on 2002/01/29 by lseiler@ma_lseiler

Preliminary version, pdf update

Change 14179 on 2002/01/29 by lseiler@ma_lseiler

Preliminary RB spec update

Change 14164 on 2002/01/29 by mpersaud@mpersaud_r400_win_tor

Initial rev

Change 14156 on 2002/01/29 by rbell@rbell_crayola_win_cvd

Added straps

Change 14155 on 2002/01/29 by rbell@rbell_crayola_win_cvd

First draft.

Change 14143 on 2002/01/29 by lseiler@ma_lseiler

Version 0.6, with depth and color compression formats

Change 14139 on 2002/01/29 by smoss@smoss_crayola_win

updated su_sc interface

Change 14138 on 2002/01/29 by markf@markf_r400_win_marlboro

R400 Area Estimate

Change 14136 on 2002/01/29 by kcorrell@kcorrell_r400_docs_marlboro

updates as a result of the design review and further discussion with designers of MH clients

Change 14135 on 2002/01/29 by jasif@jasif_r400_win_tor

Added InterModel Bus Interface Spec.

Change 14124 on 2002/01/29 by scroce@scroce_r400_win_marlboro

Some additional touches

Change 14115 on 2002/01/29 by bryans@bryans_crayola_win_cvd

Update for week of Jan 28

Change 14113 on 2002/01/29 by hartogs@fl_hartogs

Updated -- still in progress.

Change 14108 on 2002/01/29 by vromaker@MA_VIC_P4

updates, new vsds

Change 14102 on 2002/01/28 by rbagley@MA_RBAGLEY_LTXP

0.1 (a) : partial reorganization and instruction updates.

Change 14098 on 2002/01/28 by semara@semara_r400_win_tor

correct the links

Change 14093 on 2002/01/28 by semara@semara_r400_win_tor

remove

Change 14089 on 2002/01/28 by semara@semara_r400_win_tor

initial release

Change 14071 on 2002/01/28 by semara@semara_r400_win_tor

remove vcd files

Change 14068 on 2002/01/28 by semara@semara_r400_win_tor

adding vga docs

Change 14057 on 2002/01/28 by smoss@smoss_crayola_win

Updated registers and functionality

Change 14055 on 2002/01/28 by rbell@rbell_crayola_win_cvd

Fixed entry function names

Change 14052 on 2002/01/28 by jhoule@MA_JHOULE

V 1.17
Changed TX_FMT_* and VTX_FMT_* to plain FMT_*

Change 14008 on 2002/01/28 by bryans@bryans_crayola_win_cvd

Initial high level description of DV environment for Video IP (DC)

Change 13999 on 2002/01/28 by gregs@gregs_r400_win_marlboro

Added current serge control circuits and more info in general.

Change 13994 on 2002/01/28 by tho@tho_r400_win

updated dv status report

Change 13993 on 2002/01/28 by jasif@jasif_r400_win_tor

Updated Joveria's status.

Change 13979 on 2002/01/28 by efong@efong_r400_win_tor

Updated Status

Change 13969 on 2002/01/28 by rbell@rbell_crayola_win_cvd

Changes

Change 13966 on 2002/01/28 by bryans@bryans_crayola_win_cvd

Add C++ test environment documents to depot

Change 13965 on 2002/01/28 by rbell@rbell_crayola_win_cvd

Added user model read/write commands

Change 13963 on 2002/01/28 by bryans@bryans_crayola_win_cvd

Update ifgen in schedule

Change 13948 on 2002/01/25 by chwang@chwang_r400_win_cvd

Added emulation coding standard and updated tutorial.

Change 13944 on 2002/01/25 by rvelez@rvelez_r400_win_tor

Changes based on 1/18 Interface Spec Review

Change 13942 on 2002/01/25 by rbell@rbell_crayola_win_cvd

More changes

Change 13940 on 2002/01/25 by beiwang@bei_mcspec

Updated the spec to include issues resolved after the spec review as well as other open issues. Most of the changes are in pink. Also updated the bus interface description.
Updated the visio drawing on address format

Change 13939 on 2002/01/25 by tho@tho_r400_win

updated test plan

Change 13933 on 2002/01/25 by scroce@scroce_r400_win_marlboro

Added info about Module Compiler support.

Change 13929 on 2002/01/25 by tho@tho_r400_win

rom_controller_model_test_plan rev 0.2

Change 13890 on 2002/01/25 by jasif@jasif_r400_win_tor

Updated BIF_RBBM interface to user new signal names. Added error checking section. Fixed mistakes in top-level diagram.

Change 13884 on 2002/01/25 by wlawless@wlawless

just saving

Change 13860 on 2002/01/25 by rbell@rbell_crayola_win_cvd

Final version

Change 13823 on 2002/01/25 by jacarey@fl_jacarey

Checkpoint Specification

Change 13817 on 2002/01/24 by rbagley@MA_RBAGLEY_LTXP

Deleting obsolete version.

Change 13816 on 2002/01/24 by rbagley@MA_RBAGLEY_LTXP

Intermediate checkin for version 0.1 of newly revised and reorganized shader programming model doc. The doc had been
correctly renamed; the previous version is here deleted.
We also delete the former assembly syntax doc from its deprecated
location.

Change 13815 on 2002/01/24 by jasif@jasif_r400_win_tor

Updated

Change 13814 on 2002/01/24 by semara@semara_r400_win_tor

bif-vga doc

Change 13812 on 2002/01/24 by vromaker@MA_VIC_P4

update

Change 13809 on 2002/01/24 by jayw@MA_JAYW

wrong filename is RB_BlendLogic.doc

Change 13808 on 2002/01/24 by jayw@MA_JAYW

more initial writings

Change 13787 on 2002/01/24 by wlawless@wlawless

just in case

Change 13761 on 2002/01/24 by jasif@jasif_r400_win_tor

Updated

Change 13739 on 2002/01/24 by jacarey@fl_jacarey

Note added to RBBM spec indicating that the RBBM_CNTL and RBBM_SOFT_RESET registers are in the IO and MMR decode spaces.

Change 13709 on 2002/01/23 by efong@efong_r400_win_tor

PLL model simulation model spec

Change 13693 on 2002/01/23 by jhoule@MA_JHOULE

Diagrams showing multiple piecewise linear equations for the mipmap weight optimization suggested.

Change 13692 on 2002/01/23 by jhoule@MA_JHOULE

Changed SQ to better reflect it's a separate block.

Change 13690 on 2002/01/23 by jhoule@MA_JHOULE

Drawing representing the texels used in a filter around a sample point.
Containt point/bilinear/4x4 filters.

Change 13689 on 2002/01/23 by jhoule@MA_JHOULE

R300 figures showing the wraping policies.

Change 13687 on 2002/01/23 by jhoule@MA_JHOULE

0.9.3:
Updated interfaces to match current specs.
Added comments on various issues.
Described LOD optimization approach.
Added diagrams.

Change 13648 on 2002/01/23 by wlawless@wlawless

new block diagrams

Change 13646 on 2002/01/23 by rbell@rbell_crayola_win_cvd

Video IP archs and confs doc

Change 13637 on 2002/01/23 by chwang@chwang_r400_win_cvd

Another fix.

Change 13624 on 2002/01/23 by nluu@nluu_r400_cnnb

- update

Change 13623 on 2002/01/23 by jacarey@fl_jacarey

Clarification on Accessing IO and Memory-Mapped registers.
These must have the same address.

Change 13617 on 2002/01/23 by nluu@nluu_r400_cnnb

- update

Change 13603 on 2002/01/23 by chwang@chwang_r400_win_cvd

Emulation tutorial.

Change 13570 on 2002/01/22 by tho@tho_r400_win

cursor & icon documents added

Change 13546 on 2002/01/22 by bryans@bryans_crayola_win_cvd

Update status 01/22/02

Change 13534 on 2002/01/22 by mzhu@mzhu_r400_win_tor

Remove CP_DMIF_read_start signal, Add more timing diagrams.

Change 13521 on 2002/01/22 by jiezhou@jiezhou_r400_win_tor

updated Jan 22

Change 13514 on 2002/01/22 by askende@andi_r400_docs

new rev 1.2

Change 13512 on 2002/01/22 by askende@andi_r400_docs

new rev.

Change 13476 on 2002/01/21 by rbeaudin@rbeaudin_r400_win_marlboro

UPDATED REGRESSION INSTRUCTIONs

Change 13474 on 2002/01/21 by askende@andi_r400_docs

new rev. 1.1

Change 13473 on 2002/01/21 by rthambim@rthambim_r400_win_tor

Initial Revision

Change 13471 on 2002/01/21 by askende@andi_r400_docs

new rev 1.1 of the shader pipe

Change 13465 on 2002/01/21 by askende@andi_r400_docs

first time check-in

Change 13449 on 2002/01/21 by vromaker@MA_VIC_P4

update

Change 13446 on 2002/01/21 by jacarey@fl_jacarey

Checkpoint Specifications

Change 13434 on 2002/01/21 by jayw@MA_JAYW

working documents on the alpha blend data path

Change 13427 on 2002/01/21 by frising@ma_frising

-Moved Usage tabs towards back. I think this makes the spec a little more manageable than the interleaved usages.
-no version change

Change 13422 on 2002/01/21 by jiezhou@jiezhou_r400_win_tor

major updated revision

Change 13421 on 2002/01/21 by jiezhou@jiezhou_r400_win_tor

spec update

Change 13397 on 2002/01/21 by jacarey@fl_jacarey

Update CP_DEBUG and CP_STAT register fields.

Change 13391 on 2002/01/21 by efong@efong_r400_win_tor

Put in Bif model document as well as the test_control_server and verilog document from HW_modelling

Change 13383 on 2002/01/21 by jacarey@fl_jacarey

1. Outputs to VGT for 2D
2. Raster Order Determination for 2D Rectangles

Change 13380 on 2002/01/21 by chwang@chwang_r400_win_cvd

Updated.

Change 13372 on 2002/01/21 by tho@tho_r400_win

upated status report

Change 13364 on 2002/01/21 by jasif@jasif_r400_win_tor

Updated status.

Change 13361 on 2002/01/21 by jacarey@fl_jacarey

Checkpoint Updates to 2D Appendix

Change 13356 on 2002/01/21 by efong@efong_r400_win_tor

Updated % done

Change 13347 on 2002/01/21 by rbell@rbell_crayola_win_cvd

update

Change 13333 on 2002/01/18 by jacarey@fl_jacarey

Update Open Issues List
Update Go/Active Assertion Control Register

Change 13332 on 2002/01/18 by jacarey@fl_jacarey

Document Double-Registered Discrete Signals for Better Chip Timing.

Change 13331 on 2002/01/18 by jacarey@fl_jacarey

Checkpoint CP Spec

Change 13327 on 2002/01/18 by tho@tho_r400_win

ROM controller model test plan

Change 13323 on 2002/01/18 by gregs@gregs_r400_win_marlboro

initial version, based on current designs.

Change 13318 on 2002/01/18 by beiwang@bei_mcspec

Things noted from the spec review.
Spec: Changes are highlighted in pink.
MCCI drawing: reduced the number of lines going into MH read bus mux

Change 13316 on 2002/01/18 by scroce@scroce_r400_win_marlboro

Added info about v2html support and new command line switches

Change 13308 on 2002/01/18 by jasif@jasif_r400_win_tor

Updated signal interface list and top level diagram.

Change 13293 on 2002/01/18 by jowang@jowang_R400_win

area estimation for Lili

Change 13281 on 2002/01/18 by khabbari@khabbari_r400_win

added columns to see if line buffer has enough bandwidth

Change 13272 on 2002/01/18 by rbeaudin@rbeaudin_r400_win_marlboro

doc update

Change 13268 on 2002/01/18 by rbell@rbell_crayola_win_cvd

Added ati_mempak doc

Change 13263 on 2002/01/18 by rvelez@rvelez_r400_win_tor

Rev 1.1

Change 13255 on 2002/01/17 by mdoggett@MA_MDOGGETT_LT

Finished integrating and cleaning out old L1 spec details.

Change 13238 on 2002/01/17 by jacarey@fl_jacarey

Updated Size for Tag Memory and FIFO

Change 13235 on 2002/01/17 by askende@andi_r400_docs

new rev.

Change 13231 on 2002/01/17 by askende@andi_r400_docs

new shader rev of the shader spec checked in.

Change 13227 on 2002/01/17 by askende@andi_r400_docs

new rev of the shader spec

Change 13226 on 2002/01/17 by askende@andi_r400_docs

first time check-in

Change 13209 on 2002/01/17 by dglen@dglen_r400_dell

Some updates

Change 13184 on 2002/01/17 by nluu@nluu_r400_cnnb

- update

Change 13179 on 2002/01/17 by nluu@nluu_r400_cnnb

- RBBMIF block specs, timing and block diagrams

Change 13167 on 2002/01/16 by nluu@nluu_r400_cnnb

- edit

Change 13160 on 2002/01/16 by rbell@rbell_crayola_win_cvd

Another draft of the spec

Change 13143 on 2002/01/16 by nluu@nluu_r400_cnnb

- update rbbmif - autoreg spec

Change 13137 on 2002/01/16 by rbeaudin@rbeaudin_r400_win_marlboro

new R400 Emulator Specification

Change 13133 on 2002/01/16 by nluu@nluu_r400_cnnb

- update action items for RBBMIF

Change 13119 on 2002/01/16 by bryans@bryans_crayola_win_cvd

Update task list for week: 01/14/01

Change 13091 on 2002/01/15 by scroce@scroce_r400_win_marlboro

Changed some features. MKTREE is now disabled by default but can be enabled on Command line. Also, incremental compile can now be disabled

Change 13085 on 2002/01/15 by jiezhou@jiezhou_r400_win_tor

delete

Change 13075 on 2002/01/15 by nluu@nluu_r400_cnnb

- move interface doc to dc/Interface directory

Change 13074 on 2002/01/15 by rbell@rbell_crayola_win_cvd

Added the System Memory model

Change 13073 on 2002/01/15 by lseiler@ma_lseiler

revision 0.3, as checked in before Christmas

Change 13069 on 2002/01/15 by jasif@jasif_r400_win_tor

Initial revision for RBBM Model Document.

Change 13066 on 2002/01/15 by wlawless@wlawless

ok

Change 13059 on 2002/01/15 by jacarey@fl_jacarey

Checkpoint Specifications

Change 13058 on 2002/01/15 by llefebvr@llefebvre_laptop_r400

redondant opcodes corrected.

Change 13057 on 2002/01/15 by llefebvr@llefebvre_laptop_r400

There was a small error in the control flow section. Checked in the spec so that Richard has a correct version to build the assembler on.

Change 13054 on 2002/01/14 by askende@andi_r400_docs

added for the first time

Change 13015 on 2002/01/14 by jayw@MA_JAYW

Fixed visio drawing, try 2

Change 13013 on 2002/01/14 by lseiler@ma_lseiler

Version 0.7

Change 13011 on 2002/01/14 by bbloemer@ma-jasonh

Final update for release 0.7

Change 13010 on 2002/01/14 by jayw@MA_JAYW

fixed typo in Figure 7: Pixel Format winin 2D Micro-Tiles
wrong numeric bit positions

Change 13007 on 2002/01/14 by beiwang@bei_mcspec

Mods to MEMCtl and MCProtocol drawings

Change 13006 on 2002/01/14 by beiwang@bei_mcspec

Modifications to open issues, wordvalid field of access bus, tile+host queues depth, synchronization over valid&allocated bits of read data buffer, etc

Change 12989 on 2002/01/14 by yvalcour@yvalcour_r400_win_marlboro

fixed format and updated section 4.

Change 12984 on 2002/01/12 by beiwang@bei_mcspec

- Drawings: Modified MCaddressformat dawing. Slight mod to MCordering. Added MCdramcmd drawing.
- Spec: Modified address format description. Modified description to MCP and added description to SW dram cmd

Change 12983 on 2002/01/11 by nluu@nluu_r400_win

Change 12981 on 2002/01/11 by bbloemer@ma-jasonh

Another step closer to release 0.7

Change 12975 on 2002/01/11 by mdoggett@MA_MDOGGETT_LT

More updates on TCB, TCD, TCM, TCO and Formats, including document reorganisation and out dated sections removed

Change 12973 on 2002/01/11 by rvelez@rvelez_r400_win_tor

Line Buffer-Scaler Interface Spec

Change 12966 on 2002/01/11 by scroce@scroce_r400_win_marlboro

Document about the Build script

Change 12960 on 2002/01/11 by jacarey@fl_jacarey

Update Tag Memory Size

Change 12949 on 2002/01/11 by jowang@jowang_R400_win

delete #2

Change 12947 on 2002/01/11 by jowang@jowang_R400_win

first draft after 1/10/02 review

Change 12905 on 2002/01/11 by jhoule@MA_JHOULE

Current document is now R400_Texture_Pipe.doc

Change 12880 on 2002/01/10 by rvelez@rvelez_r400_win_tor

LB - Composite Pipe Interface Spec

Change 12878 on 2002/01/10 by jiezhou@jiezhou_r400_win_tor

initial release

Change 12840 on 2002/01/10 by frising@ma_frising

Closed issue 1.) about mapping of signed normalized vertex data.
No version change.

Change 12839 on 2002/01/10 by jacarey@fl_jacarey

Add diagram

Change 12827 on 2002/01/10 by rbell@rbell_crayola_win_cvd

Another draft.

Change 12826 on 2002/01/10 by rbell@rbell_crayola_win_cvd

Additions

Change 12824 on 2002/01/10 by jacarey@fl_jacarey

Fix Internal Bus Write Transaction

Change 12809 on 2002/01/10 by jacarey@fl_jacarey

Update RBBM Spec and Top-Level
1. CPQ_DATA_SWAP
2. NQ_WAIT_UNTIL

Change 12795 on 2002/01/10 by jacarey@fl_jacarey

Diagram Updates

Change 12786 on 2002/01/10 by jasif@jasif_r400_win_tor

Add Joveria's tasks.

Change 12774 on 2002/01/10 by jacarey@fl_jacarey

Update diagrams

Change 12755 on 2002/01/09 by jacarey@fl_jacarey

Checkpoint Specifications

Change 12749 on 2002/01/09 by jhoule@MA_JHOULE

Put SQ in bold to indicate a separate block (beurk).

Change 12748 on 2002/01/09 by jhoule@MA_JHOULE

Updated TC interfaces.
SP<--TC interface described.
Pre-filter FIFO more complete.
Minor corrections here and there.

Change 12728 on 2002/01/09 by dglen@dglen_r400_dell

Updated OpenGL and started section on control of display pipe

Change 12723 on 2002/01/09 by mzhu@mzhu_r400_win_tor

DMIF-CP interface spec

Change 12708 on 2002/01/09 by llefebvr@llefebvre_laptop_r400

new revision of the sequencer spec v1.6

Change 12704 on 2002/01/09 by rbell@rbell_crayola_win_cvd

First draft of the DMIF simulation model spec.

Change 12691 on 2002/01/09 by nluu@nluu_r400_win

- update action items for rbbmif

Change 12690 on 2002/01/09 by yvalcour@yvalcour_r400_win_marlboro

add mc/mh test plan to correct location.

Change 12689 on 2002/01/09 by nluu@nluu_r400_win

- rename doc

Change 12677 on 2002/01/09 by jacarey@fl_jacarey

Update CP_RBIU Interfaces

Change 12667 on 2002/01/09 by nluu@nluu_r400_win

- action items for RBBMIF

Change 12661 on 2002/01/09 by rbeaudin@rbeaudin_r400_win_marlboro

update golden image release procedure

Change 12657 on 2002/01/09 by rbeaudin@rbeaudin_r400_win_marlboro

added regression information

Change 12631 on 2002/01/09 by rbeaudin@rbeaudin_r400_win_marlboro

new doc stuff

Change 12624 on 2002/01/08 by bbloemer@ma-jasonh

Partial Update for rev 0.6

Change 12621 on 2002/01/08 by rvelez@rvelez_r400_win_tor

DCC Interface-Client Spec

Change 12618 on 2002/01/08 by nluu@nluu_r400_win

- name change

Change 12612 on 2002/01/08 by frising@ma_frising

fixed a spelling typo

Change 12611 on 2002/01/08 by frising@ma_frising

v.1.16
-remove cubemap opcode and add cubemap to DIM field.
-added a DIM_3D field to select between 3Da and 3Db sizes.
-replace references to state with constant
-rename base_offset to base_address
-rename sec_offset to mip_address

Change 12604 on 2002/01/08 by jacarey@fl_jacarey

Update Strobes and Data from Internal CP Clients to RBIU

Change 12591 on 2002/01/08 by nluu@nluu_r400_win

- revised docs

Change 12589 on 2002/01/08 by dglen@dglen_r400_dell

Updated to current plan

Change 12586 on 2002/01/08 by jacarey@fl_jacarey

Checkpoint Updates for Registers

Change 12579 on 2002/01/08 by jiezhou@jiezhou_r400_win_tor

initial release

Change 12576 on 2002/01/08 by jiezhou@jiezhou_r400_win_tor

initial release

Change 12568 on 2002/01/07 by jiezhou@jiezhou_r400_win_tor

Initial release

Change 12560 on 2002/01/07 by vromaker@MA_VIC_P4

added info on remapping tables

Change 12557 on 2002/01/07 by jacarey@fl_jacarey

Checkpoint Unit Specs

Change 12527 on 2002/01/07 by mmantor@mmantor_r400

added visio files for the Hardware state management document

Change 12509 on 2002/01/07 by nluu@nluu_r400_win

- initial revision

Change 12488 on 2002/01/07 by jacarey@fl_jacarey

Clarify cp_rbiu register read bus.

Change 12486 on 2002/01/07 by jacarey@fl_jacarey

Update CP and RBBM Interfaces
RBBM can access up to 128K Bytes in the register space.

Change 12481 on 2002/01/07 by jacarey@fl_jacarey

Updated Register Fields in CP Spec

Change 12472 on 2002/01/07 by frising@ma_frising

Update with OGL feedback.

Change 12464 on 2002/01/04 by rthambim@rthambim_r400_win_tor

Fixed syntax.

Change 12462 on 2002/01/04 by rthambim@rthambim_r400_win_tor

intial release

Change 12459 on 2002/01/04 by vromaker@MA_VIC_P4

added details to gpr allocation section

Change 12458 on 2002/01/04 by rbell@rbell_crayola_win_cvd

ROM Controller model spec.

Change 12444 on 2002/01/04 by kcorrell@kcorrell_r400_docs_marlboro

a few more details on some of the hardware aspects

Change 12433 on 2002/01/04 by wlawless@wlawless

nothing

Change 12423 on 2002/01/04 by frising@ma_frising

Fixed a typo

Change 12421 on 2002/01/04 by jhoule@MA_JHOULE

Fixed typos (didn't even change version number)

Change 12416 on 2002/01/04 by frising@ma_frising

Added features and caveats page. Resolved many of the open questions.

Change 12412 on 2002/01/04 by mdoggett@MA_MDOGGETT_LT

Added L2 data ready signal, removed TCB to TCM signal.

Change 12406 on 2002/01/04 by mdoggett@MA_MDOGGETT_LT

New version of TC spec. Updated discriptions and L1 and L2 access and decompression diagrams.

Change 12395 on 2002/01/04 by rbell@rbell_crayola_win_cvd

Added Toronto DV status

Change 12335 on 2002/01/03 by llefebvr@llefebvre_laptop_r400

backup of the spec

Change 12333 on 2002/01/03 by jacarey@fl_jacarey

Updates: RBIU, RCIU, and RTEE Sub-units

Change 12332 on 2002/01/03 by jacarey@fl_jacarey

Miscellaneous Updates to RTEE Diagram

Change 12331 on 2002/01/03 by vromaker@MA_VIC_P4

fixed links (hopefully)

Change 12330 on 2002/01/03 by jacarey@fl_jacarey

Add Internal Read Interfaces to RBIU Design

Change 12318 on 2002/01/02 by jacarey@fl_jacarey

Update Memory Interface Unit Diagram

Change 12317 on 2002/01/02 by jacarey@fl_jacarey

Update cp_rciu Diagram

Change 12315 on 2002/01/02 by jacarey@fl_jacarey

Checkpoint

Change 12313 on 2002/01/02 by scroce@scroce_r400_win_marlboro

Information about dk system

Change 12310 on 2002/01/02 by vromaker@MA_VIC_P4

HW Spec

Change 12302 on 2002/01/02 by wlawless@wlawless

just didn't want to lose my edits

Change 12297 on 2002/01/02 by smorein@smorein_r400

adding updated docs to final resting place

Change 12292 on 2001/12/31 by rbeaudin@rbeaudin_r400_win_marlboro

Moved compliation of mc.cpp to top of DLL so that module can use DLL imports and exports.

Added definitions for exporting Ferret functions.

Change 12291 on 2001/12/31 by jiezhou@jiezhou_r400_win_tor

updated

Change 12290 on 2001/12/31 by jiezhou@jiezhou_r400_win_tor

updated

Change 12286 on 2001/12/31 by jiezhou@jiezhou_r400_win_tor

Initial release

Change 12280 on 2001/12/28 by frising@ma_frising

Official v. 1.14. Add tables describing L2 interactions with texture, vertex, gamma, multicycle. Lots more clean up and clarifications. Started an open questions tab.

Change 12277 on 2001/12/28 by pmitchel@pmitchel_r400_win_marlboro

documenting partial release procedure

Change 12269 on 2001/12/28 by frising@ma_frising

another checkpoint

Change 12268 on 2001/12/28 by jiezhou@jiezhou_r400_win_tor

updated

Change 12266 on 2001/12/27 by frising@ma_frising

This version of 1.14 is a checkpoint.

Change 12259 on 2001/12/27 by jowang@jowang_R400_win

test images

Change 12256 on 2001/12/27 by rbeaudin@rbeaudin_r400_win_marlboro

fixed error underlines

Change 12222 on 2001/12/24 by rbell@rbell_crayola_win_cvd

Added Perforce and building libs

Change 12221 on 2001/12/24 by semara@semara_r400_win_tor

adding ffirst release to the VGA display interface document
updating the bif interface documents

Change 12220 on 2001/12/24 by jiezhou@jiezhou_r400_win_tor

Initial release

Change 12219 on 2001/12/24 by jiezhou@jiezhou_r400_win_tor

Initial release

Change 12218 on 2001/12/24 by jowang@jowang_R400_win

supports YCbCr and regression testing

Change 12161 on 2001/12/21 by jacarey@fl_jacarey

Check-In for the Holidays

Change 12136 on 2001/12/21 by smorein@smorein_r400

added initial rom spec

Change 12121 on 2001/12/21 by jacarey@fl_jacarey

Update

Change 12089 on 2001/12/20 by lseiler@ma_travel_micro

Updated micro-tiling format and updates to the depth/stencil and multi-sample color formats.

Change 12088 on 2001/12/20 by jacarey@fl_jacarey

Checkpointing Specifications

Change 12085 on 2001/12/20 by jacarey@fl_jacarey

Fix Signals on RBBM Top Level diagram

Change 12080 on 2001/12/20 by jhoule@MA_JHOULE

3Db format: changed max value from 12 to 11 (saving 1 bit in tag helps TC logic to meet area)

Change 12073 on 2001/12/19 by frising@ma_frising

remove this crusty old thing.

Change 12058 on 2001/12/19 by frising@ma_frising

Added some more notes on alignment constraints + misc cleanup. Removed SAD opcodes.

Change 12053 on 2001/12/19 by semara@semara_r400_win_tor

update the directory content

Change 12052 on 2001/12/19 by semara@semara_r400_win_tor

update

Change 12047 on 2001/12/19 by jhoule@MA_JHOULE

Wrong opcode order in TFetch instruction usage

Change 12044 on 2001/12/19 by jhoule@MA_JHOULE

Changed opcode order.
Offsets are now 26 bits (64-byte aligned).
Comments added (LOD equation, DWORD computation)
Footer and common header added to keep track when printing.

Change 12026 on 2001/12/19 by semara@semara_r400_win_tor

first release

Change 12025 on 2001/12/19 by semara@semara_r400_win_tor

using # of pix to interface to the display

Change 12024 on 2001/12/19 by semara@semara_r400_win_tor

first release

Change 12023 on 2001/12/19 by semara@semara_r400_win_tor

add the files for first relaease

Change 12022 on 2001/12/19 by semara@semara_r400_win_tor

update the block diagram

Change 12016 on 2001/12/19 by semara@semara_r400_win_tor

update top level diagram

Change 12014 on 2001/12/19 by semara@semara_r400_win_tor

vga top level block diagram first release

Change 12012 on 2001/12/19 by jowang@jowang_R400_win

moved into Scaler directory

Change 12010 on 2001/12/19 by jowang@jowang_R400_win

initial revision

Change 12006 on 2001/12/19 by jowang@jowang_R400_win

initial revision

Change 12004 on 2001/12/19 by jowang@jowang_R400_win

initial design: 1-D separable filters, scaling

Change 12000 on 2001/12/19 by jowang@jowang_R400_win

initial

Change 11999 on 2001/12/19 by lseiler@ma_lseiler

File check-in prior to vacation

Change 11970 on 2001/12/18 by jowang@jowang_R400_win

initial design: 1-D filters for scaling

Change 11953 on 2001/12/18 by paulv@MA_PVELLA

Minor fix.

Change 11952 on 2001/12/18 by paulv@MA_PVELLA

Modifications to MHS to reflect new changes/fixes. Some other minor fixes throughout spec (mainly in tables).

Change 11925 on 2001/12/18 by rbell@rbell_crayola_win_cvd

Updated doc

Change 11923 on 2001/12/18 by jacarey@fl_jacarey

Small Text Character Mapping Diagram

Change 11921 on 2001/12/18 by paulv@MA_PVELLA

Updated block diagram to reflect recent MH (and MC interface) changes.

Change 11919 on 2001/12/18 by rbell@rbell_crayola_win_cvd

Added docs for simulation

Change 11915 on 2001/12/18 by scroce@scroce_r400_win_marlboro

chmod the .cshrc

Change 11914 on 2001/12/18 by scroce@scroce_r400_win_marlboro

Remember to set P4CLIENT in UNIX setup

Change 11911 on 2001/12/18 by scroce@scroce_r400_win_marlboro

Reminded people to set $EDITOR if they do not like vi

Change 11895 on 2001/12/17 by askende@andi_r400_docs

new updates of the spec with regards to ALU instruction word definition,
scalar opcode list and the hardware definition of the scalar unit.

Change 11885 on 2001/12/17 by jacarey@fl_jacarey

Updates to CP MIU Diagram

Change 11854 on 2001/12/17 by jacarey@fl_jacarey

Misc.

Change 11839 on 2001/12/17 by jacarey@fl_jacarey

Interface Update for RCIU

Change 11833 on 2001/12/17 by llefebvr@llefebvre_laptop_r400

backup

Change 11824 on 2001/12/17 by lseiler@ma_lseiler

PDF file for rev 0.4a RB register spec

Change 11820 on 2001/12/17 by rbeaudin@rbeaudin_r400_win_marlboro

fixed reference to regression test

Change 11819 on 2001/12/17 by rbeaudin@rbeaudin_r400_win_marlboro

fixed typo

Change 11810 on 2001/12/15 by frising@ma_frising

v. 1.11 Lots of misc clean-up, comments, and formatting.

Change 11793 on 2001/12/14 by jhoule@MA_JHOULE

Changed filetype for exclusive check-out.

Change 11792 on 2001/12/14 by jhoule@MA_JHOULE

New diagrams used in the TP spec.

Change 11791 on 2001/12/14 by jhoule@MA_JHOULE

Removed duplicate of top-level diagram.
Removed Addressing step (now part of TC).
Changed features.
Added SP_TP diagram.
Changed filetype to exclusive open.
Rewrote Blending description (now contains Mark's LERPs with precision information).
Added Special Operations section for weird opcodes (Noise/Shadow/SADs/...)

Change 11790 on 2001/12/14 by jhoule@MA_JHOULE

Changed formats cell height (printing issues).
New SIZE (with DIM in same DWORD) ==> 3da=10:10:10, 3db=12:12:4lg
Added FETCH_VALID_ONLY for culling invalid pixels fetches (important for dependent fetches).

Change 11785 on 2001/12/14 by lseiler@ma_lseiler

RB registers with review comments and preliminary register names

Change 11754 on 2001/12/14 by rbeaudin@rbeaudin_r400_win_marlboro

fix doc

Change 11752 on 2001/12/14 by wlawless@wlawless

Update Tile Logic section

Change 11747 on 2001/12/14 by mpersaud@mpersaud_r400_win_tor

Initial Revision

Change 11728 on 2001/12/13 by lseiler@ma_lseiler

Register update for review

Change 11686 on 2001/12/13 by kcorrell@KCORRELL

update to version 6.3 of Memory Hub.doc

Change 11665 on 2001/12/13 by kcorrell@KCORRELL

updated RB - MC and MH - MC interface tables

Change 11540 on 2001/12/11 by jhoule@MA_JHOULE

Changed filetype (no multiple opens)

Change 11535 on 2001/12/11 by jhoule@MA_JHOULE

Now 4 DEGAMMA values.
Better DWORD equation (all automatic).
Added BORDER_SIZE for border texels.
Moved SIGNED_RF_MODE before NUM_FORMAT.

Change 11528 on 2001/12/11 by scroce@scroce_r400_win_marlboro

Trying to make Doc easier to follow.

Change 11503 on 2001/12/11 by llefebvr@llefebvre_laptop_r400

Version 1.5 of the sequencer spec. See Revision changes of the document for details.

Change 11484 on 2001/12/10 by scroce@scroce_r400_win_marlboro

Clarified a few commands

Change 11479 on 2001/12/10 by askende@andi_r400_docs

new revision

Change 11452 on 2001/12/10 by scroce@scroce_r400_win_marlboro

Added the actual command lines to use the client template in UNIX

Change 11443 on 2001/12/10 by llefebvr@llefebvre_laptop_r400

backup. Updated the constant memory management section by copying stuff from the R400 state management document by Mike Mantor.

Change 11411 on 2001/12/07 by kryan@kryan_r400_win_marlboro

Updated sanity test instructions to use perf_test.cpp.

Change 11401 on 2001/12/07 by jhoule@MA_JHOULE

Minor modifs.

Change 11399 on 2001/12/07 by jhoule@MA_JHOULE

Added usage tables.
Minor modifications (e.g. no more WHICH_DWORD since it uses Z instead but isn't there a problem for 3D hi-color textures?).

Change 11393 on 2001/12/07 by llefebvr@llefebvre_laptop_r400

backup.

Change 11337 on 2001/12/06 by paulv@MA_PVELLA

Made corrections/updates to MHS. Added new subsection (L2 Cache Invalidate Arbiter).

Change 11336 on 2001/12/06 by paulv@MA_PVELLA

Added Tags (I/O) and Sends (output). Changed buffer sizes. GMB = GRB.

Change 11335 on 2001/12/06 by paulv@MA_PVELLA

Changed instances GMB to GRB.

Change 11329 on 2001/12/06 by wlawless@wlawless

new RB to MC interface

Change 11319 on 2001/12/06 by paulv@MA_PVELLA

Minor modification (mostly to DISP functional block).

Change 11306 on 2001/12/06 by llefebvr@llefebvre_laptop_r400

New revision of the sequencer spec.

Change 11287 on 2001/12/06 by ctaylor@fl_ctaylor_r400_win_marlboro

Added Scan Converter Spec to P4

Change 11271 on 2001/12/06 by mpersaud@mpersaud_r400_win_tor

Added to depot - Needs major work

Change 11264 on 2001/12/05 by kcorrell@KCORRELL

updated to include top level description of GART and new MC interface.

Change 11241 on 2001/12/05 by rbeaudin@rbeaudin_r400_win_marlboro

fixed client spec

Change 11229 on 2001/12/05 by frising@ma_frising

Move TX/VTX const+inst to tp dir.

Change 11227 on 2001/12/05 by jhoule@MA_JHOULE

Renamed file for better consistency.

Change 11226 on 2001/12/05 by llefebvr@llefebvre_laptop_r400

Updated the register spec.

Change 11193 on 2001/12/05 by scroce@scroce_r400_win_marlboro

Fixed Capitalization issues in perforce client names

Change 11191 on 2001/12/05 by jacarey@fl_jacarey

Context Allocation Flags

Change 11185 on 2001/12/04 by rbeaudin@rbeaudin_r400_win_marlboro

added more emulator documentation

Change 11096 on 2001/12/03 by mmantor@mmantor_r400

temp for backup purposes only

Change 11095 on 2001/12/03 by mmantor@mmantor_r400

error in placement

Change 11093 on 2001/12/03 by mmantor@mmantor_r400

Preliminary for back up only at current time

Change 11072 on 2001/12/03 by pmitchel@pmitchel_r400_win_marlboro

change tx to tp for consistency

Change 11070 on 2001/12/03 by pmitchel@pmitchel_r400_win_marlboro

adding directories to depot

Change 11063 on 2001/12/03 by pmitchel@pmitchel_r400_win_marlboro

delete

Change 11061 on 2001/12/03 by pmitchel@pmitchel_r400_win_marlboro

mv doc_lib/parts to doc_lib/design/blocks

Change 11056 on 2001/12/03 by wlawless@wlawless

version 0.4

Change 11055 on 2001/12/03 by jacarey@fl_jacarey

Submit so directory can be moved.

Change 11052 on 2001/12/03 by hartogs@fl_hartogs

Interim check-in for move.

Change 11049 on 2001/12/03 by kcorrell@KCORRELL

temporary checkin to allow Paul to move things around again

Change 11048 on 2001/12/03 by llefebvr@llefebvre_laptop_r400

submited for Paul to move stuff around again.

Change 11047 on 2001/12/03 by askende@andi_r400_docs

more updates

Change 11006 on 2001/11/30 by scroce@scroce_r400_win_marlboro

Addded some NT specific fixes

Change 11004 on 2001/11/30 by scroce@scroce_r400_win_marlboro

Clarified some of the directions

Change 11002 on 2001/11/30 by pmitchel@pmitchel_r400_win_marlboro

moving to r400/parts_lib/

Change 10988 on 2001/11/30 by semara@semara_r400_win_tor

add the vga doc to r400/doc_lib/parts/vga

Change 10974 on 2001/11/30 by jacarey@fl_jacarey

CP's General Purpose DMA Engine

Change 10956 on 2001/11/29 by dglen@dglen_r400_dell

Updated templates

Change 10942 on 2001/11/29 by hartogs@fl_hartogs

Updated the section for the VGT to Shader interfaces.

Change 10936 on 2001/11/29 by jacarey@fl_jacarey

Checkpoint CP Specifications

Change 10889 on 2001/11/29 by jacarey@fl_jacarey

Updates to CP Memory Interface Unit Diagram

Change 10879 on 2001/11/29 by jacarey@fl_jacarey

Baseline 2D Appendix for CP Spec (Separate Document)

Change 10852 on 2001/11/28 by scroce@scroce_r400_win_marlboro

Added some more info on perforce setup

Change 10832 on 2001/11/28 by jacarey@fl_jacarey

Diagram of 2D Surface Definitions

Change 10810 on 2001/11/27 by pmitchel@pmitchel_r400_win_marlboro

mv to doc_lib/parts/sp

Change 10809 on 2001/11/27 by askende@andi_r400_docs

new rev of the spec.

Change 10807 on 2001/11/27 by pmitchel@pmitchel_r400_win_marlboro

mv to doc_lib/parts

Change 10806 on 2001/11/27 by hartogs@fl_hartogs

Check-in for relocation.

Change 10791 on 2001/11/27 by hartogs@fl_hartogs

Requested check-in. The document is in a interim state.

Change 10784 on 2001/11/27 by pmitchel@pmitchel_r400_win_marlboro

mv to doc_lib/parts

Change 10774 on 2001/11/27 by llefebvr@llefebvre_laptop_r400_emu

opened the files with the wrong client

Change 10773 on 2001/11/27 by pmitchel@pmitchel_r400_win_marlboro

dlete

Change 10772 on 2001/11/27 by pmitchel@pmitchel_r400_win_marlboro

mv to doc_lib/parts

Change 10771 on 2001/11/27 by pmitchel@pmitchel_r400_win_marlboro

mv to doc_lib/parts

Change 10770 on 2001/11/27 by pmitchel@pmitchel_r400_win_marlboro

change filetype to +l

Change 10769 on 2001/11/27 by pmitchel@pmitchel_r400_win_marlboro

mv to doc_lib/parts

Change 10767 on 2001/11/27 by pmitchel@pmitchel_r400_win_marlboro

mv to doc_lib/parts

Change 10766 on 2001/11/27 by pmitchel@pmitchel_r400_win_marlboro

mv to doc_lib/parts

Change 10765 on 2001/11/27 by lseiler@ma_lseiler

Render Backend: temporary version, not a complete release

Change 10764 on 2001/11/27 by lseiler@ma_lseiler

Memory Format v0.4: intermediate form, not a complete release

Change 10763 on 2001/11/27 by pmitchel@pmitchel_r400_win_marlboro

mv to doc_lib/parts

Change 10752 on 2001/11/27 by scroce@scroce_r400_win_marlboro

Fixed up some menu issues

Change 10745 on 2001/11/27 by ctaylor@fl_ctaylor

Updated SC->SU interface and a few words on performance.

Change 10738 on 2001/11/27 by pmitchel@pmitchel_r400_win_home

creation of spec areas

Change 10736 on 2001/11/27 by lseiler@ma_lseiler

Multisample: spec, preliminary version that needs lots of changes

Change 10732 on 2001/11/27 by pmitchel@pmitchel_r400_win_home

rename into parts/cp & parts/rbbm

Change 10717 on 2001/11/26 by jacarey@fl_jacarey

Check-In for Moving of Document Area

Change 10705 on 2001/11/26 by pmitchel@pmitchel_r400_win_marlboro

another rename to match r300

Change 10699 on 2001/11/26 by pmitchel@pmitchel_r400_win_marlboro

doing rename properly

Change 10698 on 2001/11/26 by pmitchel@pmitchel_r400_win_marlboro

fix mistake

Change 10697 on 2001/11/26 by pmitchel@pmitchel_r400_win_marlboro

recover

Change 10695 on 2001/11/26 by pmitchel@pmitchel_r400_win_marlboro

rename

Change 10693 on 2001/11/26 by llefebvr@llefebvre_laptop_r400

closing for Paul to move files around

Change 10691 on 2001/11/26 by pmitchel@pmitchel_r400_win_marlboro

rename "blocks" to "parts_lib"

Change 10676 on 2001/11/26 by llefebvr@llefebvre_laptop_r400

changed the file type to binary and locked

Change 10675 on 2001/11/26 by llefebvr@llefebvre_laptop_r400

new spin on the other documents regarding the sequencer spec

Change 10674 on 2001/11/26 by llefebvr@llefebvre_laptop_r400

new spin on the sequencer spec

Change 10673 on 2001/11/26 by pmitchel@pmitchel_r400_win_marlboro

test

Change 10668 on 2001/11/26 by paulv@MA_PVELLA

Minor changes/fixes.

Change 10667 on 2001/11/26 by paulv@MA_PVELLA

Major revision of block diagram.

Change 10666 on 2001/11/26 by scroce@scroce_r400_win_marlboro

Added info regarding the toold menu in p4win

Change 10665 on 2001/11/26 by paulv@MA_PVELLA

Updated and fixed MHS section.

Change 10644 on 2001/11/26 by pmitchel@pmitchel_r400_win_marlboro

deleting

Change 10632 on 2001/11/26 by jacarey@fl_jacarey

Checkpoint CP Specifications

Change 10501 on 2001/11/21 by askende@andi_r400_docs

opcode update

Change 10500 on 2001/11/21 by askende@andi_r400_docs

updated a couple of opcodes

Change 10489 on 2001/11/21 by kcorrell@KCORRELL

update of architectural description

Change 10396 on 2001/11/20 by rbeaudin@rbeaudin_r400_win_marlboro

new directions

Change 10365 on 2001/11/20 by pmitchel@pmitchel_r400_win_marlboro

adding file

Change 10333 on 2001/11/19 by pmitchel@pmitchel_r400_win_marlboro

release

Change 10318 on 2001/11/19 by rbeaudin@rbeaudin_r400_win_marlboro

more reg release changes

Change 10294 on 2001/11/19 by pmitchel@pmitchel_r400_win_marlboro

checkpoint

Change 10281 on 2001/11/19 by rbeaudin@rbeaudin_r400_win_marlboro

more instructions for changing block files

Change 10255 on 2001/11/19 by kcorrell@KCORRELL

Initial revision

Change 10254 on 2001/11/19 by kcorrell@KCORRELL

AIC block diagram initial revision

Change 10253 on 2001/11/19 by kcorrell@KCORRELL

update

Change 10199 on 2001/11/16 by jacarey@fl_jacarey

Checkpoint CP Specs

Change 10197 on 2001/11/16 by mdoggett@MA_MDOGGETT

friday check in

Change 10157 on 2001/11/16 by jacarey@fl_jacarey

RBBM Version 0.04 Specification Release.

Change 10156 on 2001/11/16 by pmitchel@pmitchel_r400_win_home

change filetype

Change 10153 on 2001/11/16 by pmitchel@pmitchel_r400_win_home

initial checkin

Change 10151 on 2001/11/16 by jacarey@fl_jacarey

Update Multi-target Slow Client Diagram

Change 10149 on 2001/11/16 by jacarey@fl_jacarey

Single Slow Client Waveform

Change 10144 on 2001/11/16 by jacarey@fl_jacarey

Update RT stream Event Engine Diagram

Change 10114 on 2001/11/16 by pmitchel@pmitchel_r400_win_marlboro

changed file types to disallow multiple edits

Change 10112 on 2001/11/16 by pmitchel@pmitchel_r400_win_marlboro

added explicit sync to release label as part of windows setup

Change 10056 on 2001/11/15 by dglen@dglen_r400_dell

Top level block diagrams

Change 10055 on 2001/11/15 by jhoule@MA_JHOULE

Contains instruction filed (both usage in commands, and meanings) as well as constant store fields.

Change 10053 on 2001/11/15 by jacarey@fl_jacarey

Add RBBM Diagrams to Perforce

Change 10042 on 2001/11/15 by jacarey@fl_jacarey

Checkpoint RBBM Specification

Change 10031 on 2001/11/15 by rbeaudin@rbeaudin_r400_win_marlboro

schedules

Change 9986 on 2001/11/15 by jacarey@fl_jacarey

Diagram for Real-Time Stream Event Engine.

Change 9981 on 2001/11/15 by mdoggett@MA_MDOGGETT

TC diagrams

Change 9979 on 2001/11/15 by mdoggett@MA_MDOGGETT

major rearrangement and integration of old texture pipe material.

Change 9967 on 2001/11/15 by mdoggett@MA_MDOGGETT_LT

no changes

Change 9941 on 2001/11/14 by dglen@dglen_r400_dell

First pass on list of required specs for Toronto

Change 9940 on 2001/11/14 by mdoggett@MA_MDOGGETT

First check-in of new TC spec. Brings together L1 access from TP spec and old TD spec, plus some additional diagrams and format information.

Change 9934 on 2001/11/14 by dglen@dglen_r400_dell

Updated paths and added section on used part of shared bus

Change 9923 on 2001/11/14 by jhoule@MA_JHOULE

Changed Wrapping/Clamping to Clamping/Wrapping for subblock naming issues.
Now at 0.8.9.

Change 9915 on 2001/11/14 by jhoule@MA_JHOULE

Blending scheme diagrams.
Modifs here and there.

Change 9910 on 2001/11/14 by dglen@dglen_r400_dell

Moved from arch/doc to devel/doc_lib/parts

Change 9903 on 2001/11/14 by jacarey@fl_jacarey

Update to RBBM Top-Level Diagram (Interfaces)

Change 9896 on 2001/11/14 by jacarey@fl_jacarey

Update RBBM Top-Level Per Interfaces

Change 9885 on 2001/11/14 by dglen@dglen_r400_dell

Moving files to doc_lib/parts from arch/doc

Change 9883 on 2001/11/14 by jacarey@fl_jacarey

Diagram of CP's Interface to Register Backbone.

Change 9882 on 2001/11/14 by jacarey@fl_jacarey

Update FIFO widths and depths for write interface.

Change 9878 on 2001/11/14 by jacarey@fl_jacarey

Update to CP's Memory Interface Unit (MIU) Diagram
1. Added Write Confirm Signal
2. Removed i's and o's from Interface Signals

Change 9872 on 2001/11/14 by jacarey@fl_jacarey

Update CP's View of Instruction Memory
Checkpoint other CP Specs

Change 9826 on 2001/11/13 by jhoule@MA_JHOULE

Put Sampling in a separate section than Walking, because it is closer to Wrapping.
Safety check-in.

Change 9739 on 2001/11/12 by paulv@MA_PVELLA

Initial release.

Change 9738 on 2001/11/12 by paulv@MA_PVELLA

Fixed any grammatical errors found and update of Read Bus Switch, including completion (?) of Tag Sequencer subsection.

Change 9737 on 2001/11/12 by paulv@MA_PVELLA

Updated with latest architectural changes.

Change 9724 on 2001/11/12 by scroce@scroce_r400_win_marlboro

Clarified some instructions.

Change 9723 on 2001/11/12 by askende@andi_r400_docs

new revision

Change 9720 on 2001/11/12 by scroce@scroce_r400_win_marlboro

Made setup doc instructions for perforce client more explicit

Change 9710 on 2001/11/12 by scroce@scroce_r400_win_marlboro

Fixed client spec naming from _unix_ to _sun_

Change 9653 on 2001/11/09 by jhoule@MA_JHOULE

Week end check-in.
Almost finished stripping out the TC stuff.

Change 9648 on 2001/11/09 by jhoule@MA_JHOULE

Caching is dotted line to indicate sharing between the 4 TPs.

Change 9646 on 2001/11/09 by jacarey@fl_jacarey

CP's View of Instruction Memory

Change 9627 on 2001/11/09 by jhoule@MA_JHOULE

Isolated TC.
Getting ready to strip it out to be plugged in TC (which Michael will check-in afterwards).

Change 9586 on 2001/11/08 by dclifton@dclifton_r400

new specs

Change 9561 on 2001/11/08 by scroce@scroce_r400_win_marlboro

Added some commands to build the emulator once things are set up

Change 9522 on 2001/11/08 by scroce@scroce_r400_win_marlboro

Document about the new DK_* Environment we will be using for R400

Change 9517 on 2001/11/08 by jacarey@fl_jacarey

Revision 0.03 for Preliminary Review

Change 9493 on 2001/11/08 by jacarey@fl_jacarey

Checkpoint Revision 0.02 of the PM4 Specification

Change 9396 on 2001/11/07 by kcorrell@KCORRELL

continuation of major update working towards review mid Nov

Change 9393 on 2001/11/07 by kcorrell@KCORRELL

updated diagram

Change 9278 on 2001/11/06 by jacarey@fl_jacarey

Update to RBBM Top-Level Diagram (Index DMA Request Path)

Change 9277 on 2001/11/06 by jacarey@fl_jacarey

Update CP Interface Diagrams.

Change 9214 on 2001/11/05 by jacarey@fl_jacarey

Checkpoint Updates to CP Spec and PM4 Spec:
1. Viz Query
2. Removal of Obsolete Draw Packets
3. Add DRAW_INDX and VIZ_QUERY packets
4. Add MPEG_INDEX packet.
5. Some Updates for RT streams

Change 9194 on 2001/11/05 by dwong@cndwong2

first draft of R400_IDCT

Change 9156 on 2001/11/05 by jacarey@fl_jacarey

Updates to CP's Memory Interface Unit Diagram

Change 9105 on 2001/11/02 by ctaylor@fl_ctaylor

First Rev of PA Top Level Spec,

VGT_Spec
CLIP/VTE Spec.

Change 9067 on 2001/11/02 by jacarey@fl_jacarey

Checkpoint CP Spec

Change 9045 on 2001/11/02 by jacarey@fl_jacarey

Diagram addition of Shader Instruction Fetch Path through RBBM.

Change 9028 on 2001/11/01 by jacarey@fl_jacarey

Checkpoint All RBBM and CP Documents and Diagrams.

Change 9022 on 2001/11/01 by jacarey@fl_jacarey

Update RCIU

Change 9014 on 2001/11/01 by jacarey@fl_jacarey

Update CP's MIU diagram per 32-bit interface POR.

Change 9001 on 2001/11/01 by jacarey@fl_jacarey

Rename CP-to-RBBM Interface

Change 8999 on 2001/11/01 by jacarey@fl_jacarey

Diagram of CP-to-RBBM Interface Unit

Change 8988 on 2001/11/01 by jacarey@fl_jacarey

Diagram of Index DMA Engine in CP for the PA.

Change 8961 on 2001/11/01 by wlawless@wlawless

.

Change 8932 on 2001/10/31 by kcorrell@KCORRELL

oops this is what was supposed to be submitted the last time...

Change 8927 on 2001/10/31 by kcorrell@KCORRELL

adding some new block diagrams and did a major reorg of the MH document

Change 8597 on 2001/10/26 by pmitchel@pmitchel_iris

fixed tabular state machine description

Change 8595 on 2001/10/26 by pmitchel@pmitchel_iris

added dff vs always description

Change 8571 on 2001/10/26 by pmitchel@pmitchel_iris

changed filetype to exclusive edit

Change 8551 on 2001/10/26 by bbloemer@ma-jasonh

Release with updates from review meetings.

Change 8404 on 2001/10/24 by mdoggett@MA_MDOGGETT

Small interface corrections and additions.

Change 8386 on 2001/10/24 by kcorrell@KCORRELL

block diagram of MHC and MHA

Change 8369 on 2001/10/24 by rbeaudin@MA_RAYB

moved location of autoreg doc

Change 8358 on 2001/10/24 by lseiler@ma_lseiler

Render Backend Registers v0.2: Various changes from one-on-one reviews

Change 8332 on 2001/10/24 by rbeaudin@MA_RAYB

more doc info

Change 8286 on 2001/10/23 by jacarey@fl_jacarey

Checkpoint CP & RBBM Requirements Matrix

Change 8283 on 2001/10/23 by jacarey@fl_jacarey

Checkpoint of All CP Specs (Not Complete)

Change 8282 on 2001/10/23 by jacarey@fl_jacarey

Checkpoint RBBM Spec:
1. Byte-Enables are now: BE
2. Clarifications on CP "non-queued" data path.

Change 8280 on 2001/10/23 by rbeaudin@MA_RAYB

added new block interfaces

Change 8256 on 2001/10/23 by paulv@MA_PVELLA

Updated DBR and RRA sections, with updates to functional and architectural descriptions, along with updates to the block diagrams. Also started Tag Sequencer subsection.

Change 8200 on 2001/10/22 by bbloemer@ma-jasonh

Some of the changes from the style guide meetings.

Change 8170 on 2001/10/22 by jacarey@fl_jacarey

Baseline Beginnings of Stream Fetcher Diagram

Change 8167 on 2001/10/22 by paulv@MA_PVELLA

Forgot to fix the DISP routing. It is done.

Change 8161 on 2001/10/22 by wlawless@wlawless

new file

Change 8153 on 2001/10/22 by jacarey@fl_jacarey

Update CP Memory Interface Unit Diagram.

Change 8146 on 2001/10/22 by paulv@MA_PVELLA

Fixed block diagram to reflect recent functional/architectural changes.

Change 8114 on 2001/10/19 by kcorrell@KCORRELL

Modified connections to AIC interface block (HMB)

Change 8111 on 2001/10/19 by jacarey@fl_jacarey

Memory Hub Interface Unit Diagram for CP.

Change 8106 on 2001/10/19 by jayw@MA_JAYW

Jay's exploration of recip

Change 8050 on 2001/10/18 by jacarey@fl_jacarey

RBBM Specification Version 0.02
Checkpoint CP Specification and PM4 Specification

Change 8011 on 2001/10/18 by mdoggett@MA_MDOGGETT

a few interface changes

Change 7974 on 2001/10/17 by paulv@MA_PVELLA

Refixed block diagram (what I had before last revision was mostly correct).

Change 7923 on 2001/10/17 by jhoule@MA_JHOULE

Removed PRINT field.

Change 7884 on 2001/10/16 by jhoule@MA_JHOULE

Added constant store and instruction store in the interface portion.

Change 7824 on 2001/10/16 by jhoule@MA_JHOULE

Cleaning up of interfaces. Removed duplicates.

Change 7809 on 2001/10/15 by paulv@MA_PVELLA

Fixed DBR and RRA sections with new I/O and logic adjustments.

Change 7808 on 2001/10/15 by paulv@MA_PVELLA

Updated block diagram to fix I/O and some logic changes.

Change 7806 on 2001/10/15 by paulv@MA_PVELLA

Updated block diagram to include AIC address and byte swap I/O.

Change 7779 on 2001/10/15 by jhoule@MA_JHOULE

Minor corrections here and there.

Change 7722 on 2001/10/12 by paulv@MA_PVELLA

Minor fix the the RRA block diagram (changed the tag info FIFO to a tag info RAM).

Change 7703 on 2001/10/12 by jacarey@fl_jacarey

Update Interrupt Registers in the CP.

Change 7684 on 2001/10/12 by jhoule@MA_JHOULE

Major reorganization of sections.
Not revision-savvy.

Change 7683 on 2001/10/12 by jhoule@MA_JHOULE

Added Output Formatter at the end (regroups into a quad, and converts to floats).

Change 7667 on 2001/10/12 by jacarey@fl_jacarey

Checkpoint write-up of interrupt generation.

Change 7645 on 2001/10/11 by paulv@MA_PVELLA

Fixed Data Bus Router (HDP client needed to be added).

Change 7631 on 2001/10/11 by jhoule@MA_JHOULE

New Block Description (reorganization underway).

Change 7615 on 2001/10/11 by paulv@MA_PVELLA

Minor fix.

Change 7611 on 2001/10/11 by lseiler@ma_lseiler

First draft of Render Backend register field list

Change 7581 on 2001/10/11 by paulv@MA_PVELLA

Updated block diagram to reflect recent changes (see spec).

Change 7580 on 2001/10/11 by paulv@MA_PVELLA

Updated Data Bus Router section.

Change 7554 on 2001/10/10 by jacarey@fl_jacarey

Updated some state PM4 packets -- Checkpoint of document.

Change 7487 on 2001/10/09 by jhoule@MA_JHOULE

Accepted all changes.
Various minor tweaks here and there (notably, issues aren't headings anymore).

Change 7461 on 2001/10/09 by jhoule@MA_JHOULE

Better add the top-level diagram if I want people to see it...

Change 7460 on 2001/10/09 by jhoule@MA_JHOULE

Added new top-level diagram.
Description of pipe in features to give a good idea of where we are heading.

Change 7420 on 2001/10/08 by jacarey@fl_jacarey

Incremental Update

Change 7418 on 2001/10/05 by paulv@MA_PVELLA

Another big update to RRA (with new understanding of subblock, functionality defined in much better detail). Also fixed DBR subblock section to reflect removal of BUFFER_FULL flags (now done in RRA).

Change 7417 on 2001/10/05 by paulv@MA_PVELLA

Updated diagram to reflect recent changes/adjustments.

Change 7411 on 2001/10/05 by jhoule@MA_JHOULE

Week end check in (minor corrections, mostly formatting).

Change 7371 on 2001/10/05 by mdoggett@MA_MDOGGETT

small changes to MC interfaces

Change 7370 on 2001/10/05 by mdoggett@MA_MDOGGETT

0.4 revision, final check in for major changes
(that's the idea at least)

Change 7361 on 2001/10/05 by jhoule@MA_JHOULE

Walkers more up-to-date.
Described a bit the samples fetching (pt, bilin, arbi).

Change 7350 on 2001/10/04 by paulv@MA_PVELLA

New block diagram for Read Return Arbitrator (RRA).

Change 7349 on 2001/10/04 by paulv@MA_PVELLA

Added block diagram to RRA section along with more documentation (functional/interface).

Change 7335 on 2001/10/04 by jhoule@MA_JHOULE

Updated the LOD computation pseudo-code.
This has to update walkers (with diagram), and the clamp logic (since it's now in float around [0,1] with clamp and mirror flags).

Change 7290 on 2001/10/04 by rbeaudin@Crayola_NT_Emu

more doc

Change 7242 on 2001/10/03 by rbeaudin@MA_RAYB

numbers library

Change 7231 on 2001/10/03 by rbeaudin@Crayola_NT_Emu

more docs

Change 7224 on 2001/10/03 by rbeaudin@MA_RAYB

new doc stuff

Change 7192 on 2001/10/02 by paulv@MA_PVELLA

Some minor updates.

Change 7185 on 2001/10/02 by rbeaudin@MA_RAYB

corrected documentation

Change 7149 on 2001/10/01 by wlawless@wlawless

nothing

Change 7102 on 2001/09/28 by paulv@MA_PVELLA

Fixed the data bus router's block diagram. Some minor document tweaks.

Change 7088 on 2001/09/28 by mdoggett@MA_MDOGGETT

More changes.

Change 7050 on 2001/09/27 by paulv@MA_PVELLA

A portion of the old Read Return Arbiter has become what is now known as the Data Bus Router. This is the block diagram.

Change 7044 on 2001/09/27 by jhoule@MA_JHOULE

Aesthetic modifications.
Table of contents now has correct number sizes!!! (hurray!)
Thanks to Mike "Vege-mate" Doggett for this...

Change 6992 on 2001/09/27 by rbeaudin@MA_RAYB

more doc stuff

Change 6989 on 2001/09/27 by rbeaudin@MA_RAYB

more doc stuff

Change 6970 on 2001/09/26 by jacarey@fl_jacarey

Lots of Update on:

1. Real-Time Stream Algorithm and Description
2. State Management Processing

Updated requirement matrix for CP.

Change 6889 on 2001/09/25 by askende@andi_docs

newest version

Change 6878 on 2001/09/25 by mdoggett@MA_MDOGGETT

major changes to document structure and content.
progress checkin before version 0.4

Change 6859 on 2001/09/24 by rbeaudin@MA_RAYB

more doc stuff

Change 6840 on 2001/09/24 by jacarey@fl_jacarey

Add CP/RBBM Requirements Matrix to Perforce

Change 6839 on 2001/09/24 by jacarey@fl_jacarey

Checkpoint On-Going Updates to Documents.

Change 6833 on 2001/09/24 by wlawless@wlawless

updated revision code

Change 6832 on 2001/09/24 by rbeaudin@MA_RAYB

doc update

Change 6790 on 2001/09/21 by llefebvr@llefebvre_laptop_r400

RE spec backup + HZ stats + SC spec backup

Change 6758 on 2001/09/20 by wlawless@wlawless

Slight update

Change 6757 on 2001/09/20 by jacarey@fl_jacarey

CP Spec:
1. Added registers for Real-Time Event Engine Control
2. Added some registers for state management control

PM4 SPec:
1. Updated text for the state management packets.
2. Updated text for synchronization packets.

Change 6739 on 2001/09/20 by rbeaudin@MA_RAYB

more documentation

Change 6724 on 2001/09/20 by rbeaudin@Crayola_NT_Emu

more documentation

Change 6529 on 2001/09/14 by ccoveney@chrisc_r400_docs

Finished this most of the way up.. may still need to correct some things but this is the fairly close to final version

Change 6478 on 2001/09/14 by rbeaudin@MA_RAYB

new doc

Change 6438 on 2001/09/13 by rbeaudin@MA_RAYB

emulator documentation

Change 6425 on 2001/09/13 by sallen@ma_sallen

Final update. Needs cons information...

Change 6317 on 2001/09/11 by jhoule@MA_JHOULE

Minor LODBias correction in LOD computation pseudo-code.

Change 6274 on 2001/09/10 by ccoveney@chrisc_r400_docs

Added description and screenshots of how to change clients in perforce

Change 6252 on 2001/09/10 by ccoveney@chrisc_r400_docs

This file has a short description of some specific details that help out with running perforce integrated with Developer Studio.

Change 6251 on 2001/09/10 by ccoveney@chrisc_r400_docs

This is the documentation file for using the new parameterized test library that's found in //depot/r400/verification/param_test

Change 6091 on 2001/09/05 by jacarey@fl_jacarey

General Checkpoint of Documentation

Change 5777 on 2001/08/28 by lseiler@ma_lseiler

Version 0.5a -- fixed typos in the bus interface tables

Change 5537 on 2001/08/21 by wlawless@wlawless

RB Hardware Design Spec

Change 5466 on 2001/08/17 by askende@andi_docs

new rev of the spec

Change 5465 on 2001/08/17 by jhoule@MA_JHOULE

Better LOD algorithm (leaner and better... now uses log2)

Change 5384 on 2001/08/15 by jhoule@MA_JHOULE

Initial check-in.
This file gives supplemental information on design decisions, notations, algorithms and various other underlying stuff.
It helps explain more, but reduce the already too big texture spec.

Change 5383 on 2001/08/15 by jhoule@MA_JHOULE

File wasn't saved (sorry).

Change 5382 on 2001/08/15 by jhoule@MA_JHOULE

Added LOD computation (with correction), samples walking (trilinear and anistropic), and texel fetching (with wrapping policy table) in the logic description.

Change 5365 on 2001/08/15 by pmitchel@pmitchel_iris

changed `define section

Change 5275 on 2001/08/13 by pmitchel@pmitchel_iris

fixed `define section

Change 5175 on 2001/08/09 by lseiler@ma_lseiler

Memory Controller, v0.5: removed autotag feature, changed bus/signal names and other stuff

Change 5141 on 2001/08/08 by pmitchel@pmitchel_test_client

add

Change 5136 on 2001/08/08 by bbloemer@ma-jasonh

More coding guidelines stuff.

Change 5129 on 2001/08/08 by pmitchel@pmitchel_test_client

test

Change 5090 on 2001/08/07 by pmitchel@pmitchel_test_client

add

Change 5049 on 2001/08/06 by pmitchel@pmitchel_test_client

additional comments on style guide

Change 5038 on 2001/08/03 by pmitchel@pmitchel_r400_docs

filled paragraphs to get rid of overly long lines

Change 5016 on 2001/08/03 by llefebvr@llefebvre_laptop_r400

setup unit revised spec

Change 4980 on 2001/08/01 by llefebvr@llefebvre_laptop_r400

new spec for SC and RE. Changed a bit the interface to encompass the fact that RE is now using the full precision normalized slopes from the SU because it takes the same number of bits than doing a per tile compression for the barycentric coordinates and it also simplifies a lot the SC.

Change 4960 on 2001/08/01 by askende@andi_docs

new rev

Change 4929 on 2001/07/31 by askende@andi_docs

a new rev

Change 4919 on 2001/07/31 by bbloemer@ma-jasonh

Added Verilog style guide stuff.

Change 4854 on 2001/07/30 by mdoggett@MA_MDOGGETT

Updates to most sections.

Change 4769 on 2001/07/26 by jacarey@fl_jacarey

Update Visio Diagrams
1. Chip Arch with RBBM
2. Read Data Path Diagram

Change 4756 on 2001/07/26 by pmitchel@pmitchel_iris

see if John Carey gets email

Change 4736 on 2001/07/26 by jacarey@fl_jacarey

Moved Original CP and RBBM Specs from ../arch/doc to ../doc_lib/chip/<unit> areas.

Change 4711 on 2001/07/25 by jacarey@fl_jacarey

Visio Diagram Used in the RBBM Specification

Change 4710 on 2001/07/25 by jacarey@fl_jacarey

Sweeping Updates to RBBM Spec:
1. Obsolete text deleted or crossed-out.
2. Questions inserted in document text.
3. Interfaces should be correct.

** Have Fun ! **

Change 4674 on 2001/07/24 by askende@andi_docs

new rev

Change 4597 on 2001/07/20 by jhoule@MA_JHOULE

Many minor corrections: some aesthetic, some (hopefully) helpful to understand more clearly, others are comments for Steve.

Change 4586 on 2001/07/20 by jhoule@MA_JHOULE

Minor changes: changed header for cleaner ones as well as Title Field (no more Top Level Spec).

Change 4359 on 2001/07/16 by askende@andi_docs

new rev

Change 4207 on 2001/07/11 by askende@andi_docs

new rev

Change 4206 on 2001/07/11 by askende@andi_docs

fixed a few typos

Change 4205 on 2001/07/11 by lseiler@ma_lseiler

More v0.2 files

Change 4200 on 2001/07/11 by lseiler@ma_lseiler

Render Backend: version 0.2 with greatly revised Paramater Buffer and Tile Logic sections

Change 4162 on 2001/07/10 by askende@andi_docs

new rev of the hardware spec

Change 4033 on 2001/07/06 by askende@andi_docs

update of the specs

Change 4010 on 2001/07/06 by smorein@smorein_r400

major texture pipe spec update- complete except for addresing logic.
Could use some more editing, and probably more block diagrams.

Change 4001 on 2001/07/05 by llefebvr@llefebvre_laptop_r400

lockin is on

Change 3994 on 2001/07/05 by llefebvr@llefebvre_laptop_r400

updated scan converter spec

Change 3988 on 2001/07/05 by jacarey@fl_jacarey

Initial Baseline from R300 plus listing of some open items.

Change 3980 on 2001/07/05 by jacarey@fl_jacarey

Baseline PM4 Packet Spec for CP from R300 (Khan)

** Note: Entire document may become obsolete or be revised depending
on architecture decisions for the CP ***

Change 3979 on 2001/07/05 by jacarey@fl_jacarey

Baseline R300's CP Test Bus Spec for R400

Change 3975 on 2001/07/05 by jacarey@fl_jacarey

Baseline RBBM Unit Spec and Test Bus Document from R300.
Included sections from the R400 RBBM Spec.doc in the ../arch/chip area.
** Document is Initial Version ONLY - Lot of work is still required **

Change 3908 on 2001/07/03 by lseiler@ma_lseiler

FrameBuf v0.3: some extra compressed and 3d formats, minor changes

Change 3871 on 2001/07/02 by mdoggett@MA_MDOGGETT

Minor updates to Memory Hub spec

Change 3680 on 2001/06/25 by lseiler@ma_lseiler

Frame Buffer Format, v0.2: complete rewrite

Change 3588 on 2001/06/21 by smorein@smorein_r400

Major update. There are still a bunch of inconsistancies (and misspellings) but enough there for people closely associated with the block to need to go through it.

Change 3585 on 2001/06/20 by askende@andi_docs

new rev

Change 3574 on 2001/06/20 by askende@andi_docs

new rev

Change 3565 on 2001/06/19 by askende@andi_docs

new rev

Change 3560 on 2001/06/19 by askende@andi_docs

new rev

Change 3558 on 2001/06/19 by askende@andi_docs

another rev

Change 3553 on 2001/06/19 by askende@andi_docs

another rev (rev.03) of the shader spec

Change 3476 on 2001/06/13 by smorein@smorein_r400

Texture pipe update

Change 3446 on 2001/06/12 by smorein@smorein_r400

simple update, need to do a more complete update

Change 3445 on 2001/06/12 by smorein@smorein_r400

added old version of chip presentation

Change 3428 on 2001/06/11 by smorein@smorein_r400

Adding a bunch of files

Change 3371 on 2001/06/08 by llefebvr@llefebvre_laptop_r400

spec backup

Change 3330 on 2001/06/07 by llefebvr@llefebvre_laptop_r400

safety backup

Change 3311 on 2001/06/06 by lseiler@ma_lseiler

Memory Controller: version 0.4, added internal interfaces and improved the Ordering Engine

Change 3141 on 2001/05/29 by smorein@smorein_r400

added memory hub spec. I am not happy with it, and this block wins the prize for most likely to be conpleatly redesigned.

Change 3138 on 2001/05/29 by askende@andi_docs

more updates to the spec

Change 3104 on 2001/05/25 by llefebvr@llefebvre_laptop_r400

new spin on RE, SC, Bary

Change 3020 on 2001/05/21 by askende@andi_docs

another revision of the shader spec

Change 3008 on 2001/05/21 by smorein@smorein_r400

partial update to texture spec

Change 2720 on 2001/05/10 by lseiler@ma_lseiler

Memory Controller rev 0.3: lots more details, including the external interface

Change 2712 on 2001/05/09 by askende@andi_docs

more updates

Change 2709 on 2001/05/09 by sallen@ma_sallen

add testenv spec

Change 2700 on 2001/05/09 by askende@andi_docs

Shader specifications

Change 2679 on 2001/05/09 by ccoveney@chrisc_r400

added some possible command descriptions and a sample header from the pp6_blend_all_param.dat file

Change 2651 on 2001/05/08 by ccoveney@chrisc_r400

proofread and updates a few things

Change 2650 on 2001/05/08 by ccoveney@chrisc_r400

This file contains the proposal for the new parameterized test methodology that will implemented for R400 verification.

Change 2649 on 2001/05/08 by llefebvr@llefebvre_laptop_r400

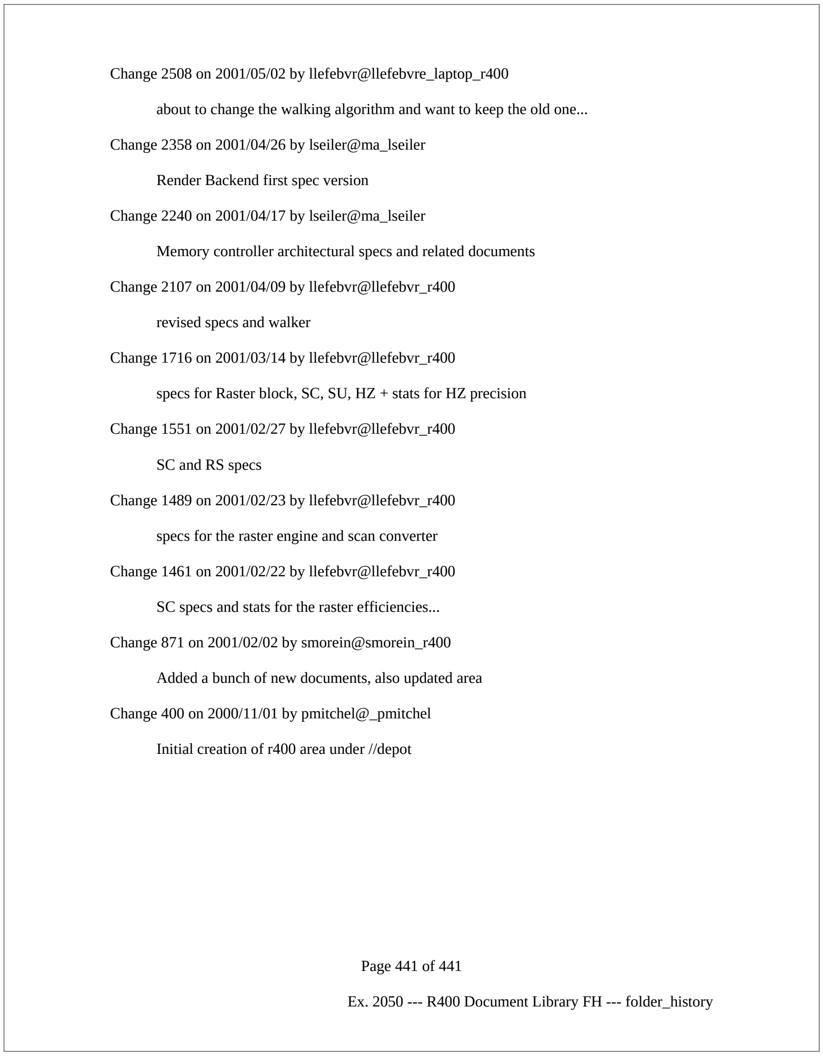
New version of the scan converter, ready for review

Change 2570 on 2001/05/04 by llefebvr@llefebvre_laptop_r400

Spin on some specs, not all complete be checked in for safety

Change 2569 on 2001/05/04 by llefebvr@llefebvre_laptop_r400

new spin on setup



Change 2508 on 2001/05/02 by llefebvr@llefebvre_laptop_r400

about to change the walking algorithm and want to keep the old one...

Change 2358 on 2001/04/26 by lseiler@ma_lseiler

Render Backend first spec version

Change 2240 on 2001/04/17 by lseiler@ma_lseiler

Memory controller architectural specs and related documents

Change 2107 on 2001/04/09 by llefebvr@llefebvr_r400

revised specs and walker

Change 1716 on 2001/03/14 by llefebvr@llefebvr_r400

specs for Raster block, SC, SU, HZ + stats for HZ precision

Change 1551 on 2001/02/27 by llefebvr@llefebvr_r400

SC and RS specs

Change 1489 on 2001/02/23 by llefebvr@llefebvr_r400

specs for the raster engine and scan converter

Change 1461 on 2001/02/22 by llefebvr@llefebvr_r400

SC specs and stats for the raster efficiencies...

Change 871 on 2001/02/02 by smorein@smorein_r400

Added a bunch of new documents, also updated area

Change 400 on 2000/11/01 by pmitchel@_pmitchel

Initial creation of r400 area under //depot

```
Change 137161 on 2003/12/12 by lseiler@lseiler_r400_win_marlboro1

        Changed sfloat<> to mfloat<>

Change 125144 on 2003/10/06 by jhoule@MA_JHOULE

        Old CacheSim modifications

Change 125143 on 2003/10/06 by jhoule@MA_JHOULE

        Old project file update

Change 121475 on 2003/09/16 by bbloemer@ma_bbloemer

        New specs.

Change 118602 on 2003/08/28 by jasony@ma-jasony-intern

        Added stop rendering and a few other features

Change 118600 on 2003/08/28 by jasony@ma-jasony-intern

        Added some comments
        brought back denorms in computegradient

Change 118379 on 2003/08/27 by jasony@ma-jasony-intern

        Clean

Change 118378 on 2003/08/27 by jasony@ma-jasony-intern

        Clean - commented out extraneous and old key functions; adapted to large texture sizes

Change 118377 on 2003/08/27 by jasony@ma-jasony-intern

        Clean - odd + even samples; moved step size calculation to aniso calculation

Change 118376 on 2003/08/27 by jasony@ma-jasony-intern

        Clean - Keeping Denorms in Subtract

Change 118373 on 2003/08/27 by jasony@ma-jasony-intern

        Added a member to QuadData to faciliate AA Correction.  Changed interpolator
precision to 23 mantissa.  Shouldn't affect existing code.

Change 118358 on 2003/08/27 by jasony@ma-jasony-intern

        Clean
```

```
Change 118348 on 2003/08/27 by jasony@ma-jasony-intern

        Fixed so that color map remains constant across texture loads (if same tex size)

Change 118337 on 2003/08/27 by jasony@ma-jasony-intern

        Checkin before final cleanup

Change 117581 on 2003/08/21 by bbloemer@ma_bbloemer

        Update.

Change 116353 on 2003/08/13 by jasony@ma-jasony-intern

        Save before playing with 8kx8k textures

Change 116158 on 2003/08/12 by jasony@ma-jasony-intern

        Save point - before cleanup and bringing in emulator changes

Change 116025 on 2003/08/12 by jasony@ma-jasony-intern

        Save before playing with precision

Change 115775 on 2003/08/11 by bbuchner@fl_bbuchner2_r400_win

        modify some simulation parameters

Change 115043 on 2003/08/05 by jasony@ma-jasony-intern

        Added the new optimizations and aniso bias

Change 114558 on 2003/08/01 by jasony@ma-jasony-intern

        Sorta working fix for magnification case

Change 114090 on 2003/07/30 by jasony@ma-jasony-intern

        Temp save - Added odd and even samples (not working yet)

Change 111136 on 2003/07/15 by jasony@ma-jasony-intern

        Added pixel shader capability

Change 110188 on 2003/07/10 by jasony@ma-jasony-intern

        cleaned up some code
```

```
Change 110154 on 2003/07/10 by jasony@ma-jasony-intern

        update the readme

Change 110153 on 2003/07/10 by jasony@ma-jasony-intern

        D3D Anisotropic filtering viewing program.

        Uses FilterSim .scn and texture files.

Change 108690 on 2003/07/01 by jasony@ma-jasony-intern

        Save point before doing Geforce FX comparison

Change 108689 on 2003/07/01 by jasony@ma-jasony-intern

        Added lod corrector

Change 107694 on 2003/06/24 by jasony@ma-jasony-intern

        Added LOD corrector (had to edit quaddata.h and walker.cpp by adding another field in
the quad structure so that the subsample is passed to to the aniso code)

Change 107693 on 2003/06/24 by jasony@ma-jasony-intern

        Added and fixed makeCylinder

Change 106801 on 2003/06/18 by jasony@ma-jasony-intern

        LOD Corrector from emulator

Change 106614 on 2003/06/17 by jasony@ma-jasony-intern

        log2 approx with 4 modes - full prec, linear approx, 5bit and 6bit table

Change 106522 on 2003/06/17 by jasony@ma-jasony-intern

        Checkin before adding LOD correction for multisampling

Change 106356 on 2003/06/16 by bbloemer@ma_bbloemer

        Added Artisan pad cells.

Change 106146 on 2003/06/13 by jasony@ma-jasony-intern

        added back old log2 approx for testing

Change 106145 on 2003/06/13 by jasony@ma-jasony-intern
```

```
        can switch to full precision with preprocessor REDUCEPREC define in header

Change 106144 on 2003/06/13 by jasony@ma-jasony-intern

        minor changes - should have no effect

Change 106143 on 2003/06/13 by jasony@ma-jasony-intern

        added emulator LOD path

        minor fixes

Change 106142 on 2003/06/13 by jasony@ma-jasony-intern

        Added makecircle and make cylinder (need to update)

Change 105665 on 2003/06/11 by jasony@ma-jasony-intern

        Modified so that emulator code will use the log2 table

Change 105423 on 2003/06/10 by jasony@ma-jasony-intern

        Working filtersim incorporating the emulator code

        important keys are

        'W' for R400 emulator aniso

        'h' for R400 full precision

        'g' for R300 full precision

Change 104930 on 2003/06/09 by jasony@ma-jasony-intern

        Checkin before hooking in EMU files

Change 104891 on 2003/06/09 by jasony@ma-jasony-intern

        Check in before precision experiments

Change 104588 on 2003/06/06 by jasony@ma-jasony-intern

        Full precision aniso tests

        1) R400: equal weights vs. weight tables [ues 'y' and 'i' keys]

        2) R300 samples vs. R400 samples (equal weight, no determinant) [use 'g' and 'u' keys]

        3) R300 vs. R400 (use 'g' and 'h' keys)
```

Change 104340 on 2003/06/05 by jasony@ma-jasony-intern

Added R300 aniso

also 'b' changes btween color mipmaps

Change 104248 on 2003/06/05 by jasony@ma-jasony-intern

Fixed and added R300 LOD and R400 LOD and Aniso calculations

added Mimap code for colored levels

Change 103884 on 2003/06/03 by jhoule@MA_JHOULE

Quick submit for Jason

Change 103799 on 2003/06/03 by jasony@ma-jasony-intern

Recover Deleted

Change 103796 on 2003/06/03 by jasony@ma-jasony-intern

deleting
figuring out perforce

Change 103793 on 2003/06/03 by jasony@ma-jasony-intern

Anisotropic Filter comparision between the R300 and R400.

The code is a modified filtersim program.

Change 99923 on 2003/05/08 by bbloemer@ma_bbloemer

klsf

Change 99291 on 2003/05/06 by bbloemer@ma_bbloemer

First GDDR4 spec.

Change 97686 on 2003/04/25 by bbloemer@ma_bbloemer

rev05 spec

Change 95532 on 2003/04/14 by bbloemer@ma_bbloemer

Originally named: JESD79-2_20030404.pdf
Release 1.0; supposedly the final for JEDEC standard.

Change 93434 on 2003/04/02 by bbloemer@ma_bbloemer

Updated 128Mb GDDR2 spec.

Added new 256Mb GDDR2 spec.

Change 88396 on 2003/03/04 by bbuchner@fl_bbuchner2_r400_win

Cache Perf Modeling -- track many more statistics.

Change 87661 on 2003/02/28 by jhoule@MA_JHOULE

Added R400 rasterization order.

Change 87226 on 2003/02/27 by smorein@stephen-moreins-Computer

correct area version of floorplan 4

Change 86554 on 2003/02/24 by smorein@stephen-moreins-Computer

fix sp order

Change 86541 on 2003/02/24 by smorein@stephen-moreins-Computer

first 4.0 floorplan

Change 82506 on 2003/02/06 by smorein@stephen-moreins-Computer

Initial docs for features r500 and DX10

Change 82306 on 2003/02/05 by ctaylor@fl_ctaylor_r400_win_marlboro

still trying

Change 82297 on 2003/02/05 by ctaylor@fl_ctaylor_r400_win_marlboro

Added code to share for cache simulation

Change 81019 on 2003/01/31 by bbloemer@ma_bbloemer

Later version of 4M x 16 TSOP.

Change 79151 on 2003/01/24 by bbloemer@ma_bbloemer

Rev075_2 from Micron.

Change 75866 on 2003/01/10 by mdoggett@MA_MDOGGETT_LT

small updates

Change 74970 on 2003/01/07 by bbloemer@ma_bbloemer

Rev 1.5

Change 74896 on 2003/01/07 by bbloemer@ma_bbloemer

Deleted duplicate copy.

Change 73431 on 2002/12/26 by smorein@stephen-moreins-Computer

cleaned up plan0.11, saved as 0.12.

Change 72690 on 2002/12/20 by smorein@stephen-moreins-Computer

first netlist 3.0 floorplan

Change 71886 on 2002/12/17 by smorein@stephen-moreins-Computer

adding floorplans

Change 71885 on 2002/12/17 by smorein@stephen-moreins-Computer

Three more floorplans

Change 71853 on 2002/12/17 by bbloemer@ma_bbloemer

Latest update - the special IP is included!

Change 71155 on 2002/12/13 by smorein@stephen-moreins-Computer

two more floorplans

Change 69955 on 2002/12/10 by smorein@stephen-moreins-Computer

Added first set of candidate floorplans

Change 67094 on 2002/11/26 by bbloemer@ma_bbloemer

DDR I spec.

Change 64756 on 2002/11/18 by bbloemer@ma_bbloemer

0288E10_draft2 version

Change 58862 on 2002/10/23 by bbloemer@ma_bbloemer

SGRAM - OLD!

Change 55651 on 2002/10/07 by bbloemer@ma_bbloemer

DDR II data sheet.

Change 51711 on 2002/09/16 by bbloemer@ma_bbloemer

Micron verilog model.

Change 51357 on 2002/09/13 by bbloemer@ma_bbloemer

Info

Change 51347 on 2002/09/13 by bbloemer@ma_bbloemer

First draft. Elpida 4Mx32 0288E10_draft2.pdf

Change 50587 on 2002/09/10 by bbloemer@ma_bbloemer

A slightly later version, sent 8/12, but still no IP. Rev 3 is latest with IP.

Change 50574 on 2002/09/10 by bbloemer@ma_bbloemer

Latest version, dated 20020916. They added tRTP.

Change 49008 on 2002/08/30 by jasony@ma_jasony

Shadow Sim instructions

Change 48247 on 2002/08/27 by lseiler@lseiler_r400_win_marlboro2

Supports testing new Zrange format

Change 46146 on 2002/08/14 by bbloemer@ma_bbloemer

GDDR III Data sheets.

Change 46137 on 2002/08/14 by bbloemer@ma_bbloemer

Added GDDR II data sheet. Like JEDEC DDR II, but has built in termination.

Change 45642 on 2002/08/13 by jasony@ma_jasony

Final Checkin

Change 45641 on 2002/08/13 by jasony@ma_jasony

Added code to facilitate plane calculations using triangle verticies.
(Does not affect previous operations)

If using ShadowSim you should comment out the precision reduction in Interpolate()

Change 45637 on 2002/08/13 by jasony@ma_jasony

Added to QuadData and Walker to facilitate plane calculation using triangle verticies

Change 45635 on 2002/08/13 by jasony@ma_jasony

Final Check In

Change 45210 on 2002/08/09 by jasony@ma_jasony

Save before code cleanup

Change 44911 on 2002/08/08 by bbloemer@ma_bbloemer

Added Samsung spec; it's really the same part as K4D263238A-GC.

New version of GDDRIII spec, but WITHOUT the new IP. Need the previous version for that.

Change 44909 on 2002/08/08 by bbloemer@ma_bbloemer

Latest version from Infineon.

Change 44908 on 2002/08/08 by bbloemer@ma_bbloemer

Fixed naming problem.

Change 44904 on 2002/08/08 by bbloemer@ma_bbloemer

Deleted file under one name and added under a better name.

Change 44660 on 2002/08/07 by jasony@ma_jasony

fixed to arbitrary shadow buffer size up to 1024

Change 44579 on 2002/08/06 by jasony@ma_jasony

z calc based on triangle verticies done

now fixing ctrl-mouse click to display slope calcs

Change 44315 on 2002/08/05 by alleng@alleng_screendiv2

Fixed data gathering problem where stencil compression data was being double counted in some cases...

Change 44309 on 2002/08/05 by jasony@ma_jasony

Convereted to calcualtions of planes from triangle verticies

save point - now fixing offset for interpolation

Change 44307 on 2002/08/05 by jasony@ma_jasony

Convereted to calcualtions of planes from triangle verticies

save point - now fixing offset for interpolation

Change 44090 on 2002/08/02 by alleng@alleng_screendiv2

Add quad mask binning

Change 44065 on 2002/08/02 by jasony@ma_jasony

Working shadow simulator

save point before figuring out what causes precision reduction error

Change 43809 on 2002/08/01 by alleng@alleng_screendiv2

Added Z and S pass/fail numbers for quads
Bunch of reformatting to make interpretation less ambiguous

Change 43557 on 2002/07/31 by jasony@ma_jasony

Working in bilenear interpolation

'y' turn on/off interpolation
'h' save precision file

Change 43487 on 2002/07/31 by alleng@alleng_screendiv2

Fixed blending bug to correctly determine RMW's
Added stencil function binning

Change 43195 on 2002/07/30 by jasony@ma_jasony

Save point

'u' increase bias
'i' decrease bias

Change 43090 on 2002/07/30 by alleng@alleng_screendiv2

Added data gathering for tile, quad, and depth cache counts during the shadow passes

Change 42948 on 2002/07/29 by alleng@alleng_screendiv2

Added logic to determine if tile was dirty
+ Added dirty flag to depth cache (Z and S)
+ Add ptr to link tile with entry in depth cache
+ Don't count flushes when the Z or S caches aren't dirty
Fixed two typos in stencil compression logic
Don't count flushes/fills when Z and S disabled

Change 42752 on 2002/07/26 by alleng@alleng_screendiv2

Added early out for tiles w/ zero pixels
Minor change to output format
Slight mod to determination of depthProcEn
Slight change to depth cache flush/fill data calculations

Change 42658 on 2002/07/26 by alleng@alleng_screendiv2

Added stencil buffer support

Change 42135 on 2002/07/24 by bbloemer@ma-jasonh

Latest rev adds special IP description as an appendix.

Change 41803 on 2002/07/22 by alleng@alleng_screendiv2

Add alpha blend stats
Add scissor functionality

Change 41600 on 2002/07/19 by llefebvr@llefebvre_laptop_r400

SQ backup.

Change 41362 on 2002/07/18 by alleng@alleng_screendiv2

Properly prevent z writes when they are masked off

Change 41351 on 2002/07/18 by jasony@ma_jasony

Save Point - modified precision controls

'(' - modify rendering precision
')' - modify shadow map/plane precision
'<' - reset shadow map/plane precision
'>' - reset rendering precision
'up'/'down' - pick operation to modify
'left'/'right' - decrease/increase precision

Change 41316 on 2002/07/18 by jasony@ma_jasony

Save Point - Looking into Precision Calculation

Added:
'g' - load precision values for calculations
'<', '>' - increase decrease Shadowfunc precision
'(', ')' - increase decrease shadowfuncAA precision

Change 41166 on 2002/07/17 by alleng@alleng_screendiv2

Don't update z if depth is disabled
Add tile and quad counters for various state settings
Reformat depth complexity output
Moved pkt->Init() to only be called for valid frame

Change 40917 on 2002/07/16 by alleng@alleng_screendiv2

+ Added pixel counts (redundant but per frame...)
+ Modified depth test to use the actual depth test function; this included beginFrame support to clear the depth buffer with the correct value at the correct time)
+ Fixed calcDepthCacheFlush/Fill routines to fall back to expanded mode when number of zplanes is > 16 and to treat stencil separate from depth

Change 40694 on 2002/07/15 by alleng@alleng_screendiv2

Added total number of tiles to hz output
Added support for stencil only pass
Split the calcDepthCacheTransfer routine into two; one for flushes and one for fills...

Change 40021 on 2002/07/12 by alleng@alleng_screendiv2

Fix frame processing (can now run multiple Doom3 frames)
Fixed bug that was mishandling EndOfFrame events
Fixed bug in previous checkin that cleared the zbuffer prematurely

Change 39936 on 2002/07/12 by jasony@ma_jasony

Changed Buffer.cpp and ShadowSim.cpp to allow for arbitrary sized shadow maps

Change 39928 on 2002/07/12 by alleng@alleng_screendiv2

Convert countZplanes() to use 64 bit mask
Added ability to pass in resX (-sx) and resY (-sy)
Many small formatting chnages
Disabled sample frame initially
Flush depth cache every frame

Change 39817 on 2002/07/11 by jasony@ma_jasony

Working ShadowAA - still need to explain ragged edges

'%' Turn on and off highlight of selected point

Change 39544 on 2002/07/10 by alleng@alleng_screendiv2

Significant changes to screendiv, including (but not limited to):

+ removing testMode
+ use 8x8 tile by default
+ rework spitQuads
+ provided mechanism to model depth cache
+ added code to gather stats on depth complexity
+ cleaned up a whole lot
+ fixed bug in countZplanes that changed baseline (break vs continue)
+ changed countZplane interface
+ isolating individual print routines within respective classes to modularize things and simplify the output

Did test this against a baselevel using d3-combat-1.bin dump file. The bug fix will force a new base line but so far the tool is reporting the same info that it did originally with a whole bunch more functionality.

Change 39478 on 2002/07/10 by jasony@ma_jasony

Save Point - Sorta working Shadow AA

Change 39477 on 2002/07/10 by bbloemer@ma-jasonh

Elpida's description of special IP. Overview only.

Change 39352 on 2002/07/10 by jasony@ma_jasony

Lets you see the multiple shadow planes in the view shadow plane mode by using the up and down cursor keys

Change 39155 on 2002/07/09 by jasony@ma_jasony

Save Pont - Rough working Shadowing with AA

-Added '$' to turn on/off AA for shadowing

Change 39106 on 2002/07/09 by jasony@ma_jasony

Save point - Shadowing working in Buffer::RenderAA (moving out to Buffer::Render)

Change 38811 on 2002/07/08 by jasony@ma_jasony

Modified to work with ShadowSim (added ls,lt,lq,lr coords to Vertex and QuadData)

Change 38810 on 2002/07/08 by jasony@ma_jasony

Working shadows

Change 37971 on 2002/07/03 by jasony@ma_jasony

Added mouse code to check points on the image

Control+mouse click sets a position (can be seen in corresponding light space)

shift+mouse click will print out screen coordinates (origin is lower left)

Change 37825 on 2002/07/03 by jhoule@MA_JHOULE

Added hooks for cam/light control.
F5: cam view
F6: light view
F7: ctrl cam
F8: ctrl light
!: Refresh light map
#: Camera mode (old '!' functionality)

Clamped position in light map to prevent crash (equivalent to clamp to last).

Change 37809 on 2002/07/03 by jasony@ma_jasony

Semi-working shadowing

Change 37645 on 2002/07/02 by jasony@ma_jasony

save point: semi working plane equation

Change 37436 on 2002/07/01 by jasony@ma_jasony

Save point - before using shadow function

Change 37383 on 2002/07/01 by jasony@ma_jasony

Save point

Change 34813 on 2002/06/18 by jasony@ma_jasony

New check-in of shadow buffer simulator (based on filtersim)

Change 34556 on 2002/06/17 by jasony@ma_jasony

sampling pattern candidates

Change 34541 on 2002/06/17 by jasony@ma_jasony

Added 3-sample options

Change 34133 on 2002/06/14 by jasony@ma_jasony

removed very old patterns

Change 34132 on 2002/06/14 by jasony@ma_jasony

Changed to allow arbitrary number of samples for anti-aliasing

Change 34129 on 2002/06/14 by jasony@ma_jasony

Recommended sampling patterns

Change 33763 on 2002/06/13 by bbloemer@ma-jasonh

M. Litt's Status.

Change 33744 on 2002/06/13 by bbloemer@ma-jasonh

Added ati GDDR-III spec update and SEC's GDDR-II spec.

Change 33609 on 2002/06/12 by jasony@ma_jasony

Changed menus

Change 33482 on 2002/06/12 by jasony@ma_jasony

updated loadSamplingPattern to accept arbitrary number of points up to 8

Change 33480 on 2002/06/12 by jasony@ma_jasony

Changed makePinWheel to take angle in degrees

Change 33456 on 2002/06/12 by jasony@ma_jasony

fixed steping

Change 33455 on 2002/06/12 by jasony@ma_jasony

changed menu

Change 33454 on 2002/06/12 by jasony@ma_jasony

changed random generation

Change 33190 on 2002/06/11 by alleng@alleng_screendiv2

Several new functions and some bug fixes (due to new parser lib updates) and logistical modifications:

+ Added 'correct' depth check for pixels rendered w/ depth test disabled; required new routine GetPrimState and some additional back pointers for HZ and ZBUFFER to get back to the Rasterizer class
+ Added coverage bins for tiles and quads
+ Modified file I/O to use the 'standard' parser lib syntax (see usage text in main() for details

Change 32959 on 2002/06/10 by jasony@ma_jasony

fixed bug in updateWalkerSamples

Change 32696 on 2002/06/07 by jasony@ma_jasony

makePinWheel

Change 32695 on 2002/06/07 by jasony@ma_jasony

Added Sampling and makePinWheel

Change 32650 on 2002/06/07 by jhoule@MA_JHOULE

Solved a few multisampling issues.
Now uses 8 samples by default.
Can now correctly blend the samples together (old scheme lost precision for small values).
Background color is now black by default.

Change 32611 on 2002/06/07 by jasony@ma_jasony

Added (makePinWheel radius number angle)

Change 32610 on 2002/06/07 by jasony@ma_jasony

fixed loading of samples

Change 32592 on 2002/06/07 by jasony@ma_jasony

Added loading of sampling patterns

Change 32385 on 2002/06/06 by jhoule@MA_JHOULE

Cleaned up for easier usage.
Added sampling pattern load hooks.

Change 32160 on 2002/06/05 by jasony@ma_jasony

No Discrepancy Function

Change 31840 on 2002/06/04 by jasony@ma_jasony

Changed to work with relative paths

Change 31839 on 2002/06/04 by jasony@ma_jasony

Fixed to work with Driectx 8.1

Change 31757 on 2002/06/04 by jasony@ma_jasony

Relevant sample patterns

Change 31756 on 2002/06/04 by jasony@ma_jasony

fixed image inversion

Change 31753 on 2002/06/04 by jasony@ma_jasony

program to generate fourier transform
required by the browser

Change 31751 on 2002/06/04 by jasony@ma_jasony

removed libtiff links

Change 31750 on 2002/06/04 by jasony@ma_jasony

Incorporated Tiff reading (this version does not need libtiff)

Change 31712 on 2002/06/04 by jasony@ma_jasony

Added search for divers.exe

Change 31620 on 2002/06/03 by jasony@ma_jasony

Changed string code

Change 31507 on 2002/06/03 by jasony@ma_jasony

Jason's antialiasing browser

Change 30945 on 2002/05/30 by bbloemer@ma-jasonh

First GDDR III spec.

Change 29844 on 2002/05/24 by bbloemer@ma-jasonh

New spec version. Original file name: JESD90-24May2002.prn.pdf

Change 29762 on 2002/05/23 by alleng@alleng_screendiv2

Added error handler to main loop to catch any errors that the Parser library (et al) may throw. The lack of a handler was apparently causing a problem when a dmp file with an error was being read.

Change 29738 on 2002/05/23 by jhoule@MA_JHOULE

Added better LOD computation schemes, with runtime control.
Added Sampler class, to control sampling position as well as ROM table to LOD correction (plane scheme).
Changed sampling pattern to something rather good.

Change 29325 on 2002/05/21 by lseiler@lseiler_r400_win_marlboro

Removed outmoded code, cleaned up some comments

Change 29309 on 2002/05/21 by lseiler@lseiler_r400_win_marlboro

Fix missing jobs.txt parameter in project file

Change 29270 on 2002/05/21 by lseiler@lseiler_r400_win_marlboro

Updated workspace and project files

Change 29236 on 2002/05/21 by lseiler@lseiler_r400_win_marlboro2

List of dump files to test

Change 29131 on 2002/05/20 by lseiler@lseiler_r400_win_marlboro

Altered to look in /r400/parser for PM4tools files

Change 29109 on 2002/05/20 by lseiler@lseiler_r400_win_marlboro2

Updated ScreenDiv2: eliminates TorontoParser to use PM4tools directory

Change 29004 on 2002/05/20 by bbloemer@ma-jasonh

Call notes.

Change 28342 on 2002/05/16 by bbloemer@ma-jasonh

Minutes.

Change 26690 on 2002/05/07 by bbloemer@ma-jasonh

Added documents.

Change 25117 on 2002/04/26 by jhoule@MA_JHOULE

Added many LOD computations schemes.
Also added various length approximations.
Hooked keyboard keys to control those.
Changed camera behaviour.

Change 24979 on 2002/04/25 by jhoule@MA_JHOULE

Spin wrongly multiplied angle by 180

Change 23311 on 2002/04/15 by bbloemer@ma-jasonh

Presentations used in telecon.

Change 23151 on 2002/04/12 by mdoggett@MA_MDOGGETT

Adding first version of Address Translate. Uses previous function types, needs to be updated for new version. Only checking it in so Ken can see it.

Change 22619 on 2002/04/10 by bbloemer@ma-jasonh

Material from April 9/10 face to face.

Change 21863 on 2002/04/04 by lseiler@lseiler_r400_win_marlboro2

Random updates

Change 21661 on 2002/04/03 by lseiler@SEILER2

Includes tile Zrange comparisons

Change 21210 on 2002/04/01 by bbloemer@ma-jasonh

Added Michael's notes.

Change 21052 on 2002/03/29 by lseiler@SEILER2

Updated with 128-bit Zplane compression ratios

Change 20653 on 2002/03/27 by jhoule@MA_JHOULE

New (hopefully, non-crashing) project files.

Change 20651 on 2002/03/27 by jhoule@MA_JHOULE

Update with various LOD computation algos.

Change 20650 on 2002/03/27 by jhoule@MA_JHOULE

Uses same jitter table for all pixels of all quads.

Change 20476 on 2002/03/26 by rbagley@MA_RBAGLEY

This has been moved to r400/doc_libs/design/chip

Change 20066 on 2002/03/21 by bbloemer@ma-jasonh

Adding history of Samsung-Elpida-ATI 3-way calls.

Change 19442 on 2002/03/18 by bbloemer@ma-jasonh

Added new memory matrix.

Change 19073 on 2002/03/15 by mdoggett@MA_MDOGGETT

Removed old L1 Tag compare.

Change 18687 on 2002/03/13 by bbloemer@ma-jasonh

Hynix presentation of 3/16/02 (not sure when actually given) and Etron data sheet.

Change 18686 on 2002/03/13 by mdoggett@MA_MDOGGETT

Fixed old L1 Miss calculation. changed flushL1 to set tagaddr to -1 instead of 0

Change 18618 on 2002/03/12 by bbloemer@ma-jasonh

Spec.

Change 18598 on 2002/03/12 by mdoggett@MA_MDOGGETT

Added simple L1Tag compare. Fixed bugs in older L1 Tag compare

Change 18597 on 2002/03/12 by mdoggett@MA_MDOGGETT

added new l1tag compare

Change 18171 on 2002/03/08 by lseiler@SEILER2

HiZ 2x2 test version

Change 16292 on 2002/02/22 by lseiler@SEILER2

Updated HiZ test results

Change 16291 on 2002/02/22 by lseiler@SEILER2

Updated for latency tests, streak tests, and cache miss effects

Change 16286 on 2002/02/22 by bbloemer@ma-jasonh

Spice simulation results for OD driver.

Change 16210 on 2002/02/21 by mdoggett@MA_MDOGGETT_LT

Updated formats. Added 2d interlace format and 3d four layer format. Added L2 read synchronisation description.

Change 16010 on 2002/02/20 by bbloemer@ma-jasonh

Elpida presentation.

Change 16001 on 2002/02/20 by bbloemer@ma-jasonh

Elpida material for next week's face to face.

Change 14507 on 2002/02/04 by jhoule@MA_JHOULE

Added dynamic filterKernel pass after rendering (only when AA).
Emu integration (unfinished).

Change 14506 on 2002/02/04 by jhoule@MA_JHOULE

Added emuWrap stuff (absolute path, though)

Change 14412 on 2002/02/01 by bbloemer@ma-jasonh

Added 18 Jan. 2002 version.

Change 14410 on 2002/02/01 by bbloemer@ma-jasonh

Added 10Sep2001 version of spec.

Change 14409 on 2002/02/01 by bbloemer@ma-jasonh

The initial 30-Nov2000 spec version.

Change 14300 on 2002/01/31 by lkang@lkang_r400_win_tor

This directory is for DC-MH interface block architecture and implementation specs

Change 14161 on 2002/01/29 by bbloemer@ma-jasonh

DRAM model documentation.

Change 14060 on 2002/01/28 by lseiler@SEILER2

Zplane charts and statistics for the 8 benchmark cases

Change 13938 on 2002/01/25 by lseiler@SEILER2

Updated with MaxMin tests

Change 13902 on 2002/01/25 by bbloemer@ma-jasonh

Data from Elpida, Samsung, ATI face to face. Moved Elpida file from Elpida directory to here.

Change 13816 on 2002/01/24 by rbagley@MA_RBAGLEY_LTXP

Intermediate checkin for version 0.1 of newly revised and reorganized shader programming model doc. The doc had been
correctly renamed; the previous version is here deleted.
We also delete the former assembly syntax doc from its deprecated
location.

Change 13768 on 2002/01/24 by bbloemer@ma-jasonh

Elpida presentation of new IP; 4 new Hynix specs, inc. > 300 MHz.;
Samsung and Nanya DRAMs > 300 Mhz.

Change 13295 on 2002/01/18 by lseiler@SEILER2

Updated with 0-, 8-, and 64-entry cache data

Change 13236 on 2002/01/17 by lseiler@ma_lseiler

Update to account for direct mapped caching on the tiles

Change 13123 on 2002/01/16 by jayw@MA_JAYW

Split out SX and RC on page and added a FIFO to SX

Change 13096 on 2002/01/15 by rbagley@MA_RBAGLEY_LT

Assembly Syntax

Change 13088 on 2002/01/15 by wlawless@wlawless

New RB estimates

Change 13066 on 2002/01/15 by wlawless@wlawless

ok

Change 13030 on 2002/01/14 by llefebvr@llefebvre_laptop_r400

updated the barycentric document

Change 12990 on 2002/01/14 by llefebvr@llefebvre_laptop_r400

2 samples now available

Change 12961 on 2002/01/11 by lseiler@ma_travel_micro

Updates made during vacation (needs to be debugged)

Change 12805 on 2002/01/10 by lseiler@SEILER2

Added table for fully covered pixels and quads

Change 12572 on 2002/01/07 by lseiler@ma_travel_micro

Updated to compute the number of Zplanes per tile and per quad

Change 12030 on 2001/12/19 by lseiler@ma_lseiler

Spreadsheets added before vacation

Change 12021 on 2001/12/19 by lseiler@SEILER2

checkins of spreadsheet result files (prior to vacation)

Change 11800 on 2001/12/14 by rbagley@MA_RBAGLEY_LT

Corrections to control flow.
Updated control flow syntax.
Misc. local corrections.
Format corrections, prior to conversion to Programming Model doc.

Change 11756 on 2001/12/14 by llefebvr@llefebvre_laptop_r400

Constant management model simulator for the R400

Change 11729 on 2001/12/13 by lseiler@SEILER2

Update to support variable precision HiZ values

Change 11560 on 2001/12/12 by rbagley@MA_RBAGLEY_LT

Interrim check-in.

Change 11553 on 2001/12/11 by rbagley@MA_RBAGLEY_LT

Extension of instruction reference to fetch instruction.
Revision of alu syntax and guide (in progress).
Miscellaneous corrections and revisions.

Change 11538 on 2001/12/11 by lseiler@SEILER2

more screendiv1 files

Change 11537 on 2001/12/11 by lseiler@SEILER2

HiZ test code (Screendiv1 for R100 and Screendiv2 for R200)

Change 11536 on 2001/12/11 by lseiler@SEILER2

Update

Change 11332 on 2001/12/06 by rbagley@MA_RBAGLEY_LT

Intermediate check-in.
Midway through updates of fetch syntax.
We also revised the register indexing.
Several other additions, including a table of resources.

Change 11124 on 2001/12/03 by rbagley@MA_RBAGLEY_LT

Corrections and improvements following Ken's reading of Dec 3.

Change 11013 on 2001/11/30 by rbagley@MA_RBAGLEY_LT

Corrections to indexing mode specification.

Change 10869 on 2001/11/28 by rbagley@MA_RBAGLEY_LT

Indexing modes description, syntax, and examples.

Interrim check-in towards version 0.4.

Change 10839 on 2001/11/28 by rbagley@MA_RBAGLEY_LT

Syntax for control flow instructions.

Began register indexing; in progress.

(Interrim check-in for version 0.4)

Change 10824 on 2001/11/27 by rbagley@MA_RBAGLEY_LT

Interrim check-in for version 0.4.

Change 10810 on 2001/11/27 by pmitchel@pmitchel_r400_win_marlboro

mv to doc_lib/parts/sp

Change 10807 on 2001/11/27 by pmitchel@pmitchel_r400_win_marlboro

mv to doc_lib/parts

Change 10806 on 2001/11/27 by hartogs@fl_hartogs

Check-in for relocation.

Change 10791 on 2001/11/27 by hartogs@fl_hartogs

Requested check-in. The document is in a interim state.

Change 10784 on 2001/11/27 by pmitchel@pmitchel_r400_win_marlboro

mv to doc_lib/parts

Change 10772 on 2001/11/27 by pmitchel@pmitchel_r400_win_marlboro

mv to doc_lib/parts

Change 10771 on 2001/11/27 by pmitchel@pmitchel_r400_win_marlboro

mv to doc_lib/parts

Change 10769 on 2001/11/27 by pmitchel@pmitchel_r400_win_marlboro

mv to doc_lib/parts

Change 10767 on 2001/11/27 by pmitchel@pmitchel_r400_win_marlboro

mv to doc_lib/parts

Change 10766 on 2001/11/27 by pmitchel@pmitchel_r400_win_marlboro

mv to doc_lib/parts

Change 10765 on 2001/11/27 by lseiler@ma_lseiler

Render Backend: temporary version, not a complete release

Change 10764 on 2001/11/27 by lseiler@ma_lseiler

Memory Format v0.4: intermediate form, not a complete release

Change 10763 on 2001/11/27 by pmitchel@pmitchel_r400_win_marlboro

mv to doc_lib/parts

Change 10745 on 2001/11/27 by ctaylor@fl_ctaylor

Updated SC->SU interface and a few words on performance.

Change 10736 on 2001/11/27 by lseiler@ma_lseiler

Multisample: spec, preliminary version that needs lots of changes

Change 10704 on 2001/11/26 by rbagley@MA_RBAGLEY_LT

Correction to export syntax.

Change 10690 on 2001/11/26 by rbagley@MA_RBAGLEY_LT

Updated export syntax. We also include a preliminary look at the export address mapping.

Change 10668 on 2001/11/26 by paulv@MA_PVELLA

Minor changes/fixes.

Change 10667 on 2001/11/26 by paulv@MA_PVELLA

Major revision of block diagram.

Change 10665 on 2001/11/26 by paulv@MA_PVELLA

Updated and fixed MHS section.

Change 10635 on 2001/11/26 by lseiler@SEILER2

final screendiv update before branching off to a separate source set

Change 10503 on 2001/11/21 by lseiler@ma_lseiler

Revised summary of frames used by tile tests

Change 10501 on 2001/11/21 by askende@andi_r400_docs

opcode update

Change 10500 on 2001/11/21 by askende@andi_r400_docs

updated a couple of opcodes

Change 10489 on 2001/11/21 by kcorrell@KCORRELL

update of architectural description

Change 10464 on 2001/11/21 by lseiler@SEILER2

Modified to eliminate pointless warning messages

Change 10431 on 2001/11/20 by rbagley@MA_RBAGLEY_LT

A first cut at the export syntax and language guide for version 0.4.

Change 10416 on 2001/11/20 by lseiler@SEILER2

Changed depth test from < to <= (second try at update)

Change 10404 on 2001/11/20 by lseiler@SEILER2

Depth test changed from < to <=

Change 10348 on 2001/11/19 by rbagley@MA_RBAGLEY_LT

Interrim check-in for version 0.4. Syntax for exports and fetches in progress.

Change 10323 on 2001/11/19 by lseiler@SEILER2

Updated to provide more HiZ information

Change 10322 on 2001/11/19 by lseiler@SEILER2

Updated to report more HiZ data (this time for real)

Change 10289 on 2001/11/19 by lseiler@SEILER2

Private version of numbers.h and numbers.cpp for screendiv -- need to change over to using the official versions in devel/cmn_lib

Change 10274 on 2001/11/19 by lseiler@ma_lseiler

update

Change 10255 on 2001/11/19 by kcorrell@KCORRELL

Initial revision

Change 10254 on 2001/11/19 by kcorrell@KCORRELL

AIC block diagram initial revision

Change 10253 on 2001/11/19 by kcorrell@KCORRELL

update

Change 10203 on 2001/11/16 by rbagley@MA_RBAGLEY_LT

Interim check-in of revision 0.4. In progress.

Change 10197 on 2001/11/16 by mdoggett@MA_MDOGGETT

friday check in

Change 10184 on 2001/11/16 by lseiler@SEILER2

various updates

Change 10182 on 2001/11/16 by lseiler@ma_lseiler

Tiling code, just before splitting off the heir directory

Change 10172 on 2001/11/16 by llefebvr@llefebvre_laptop_r400

chikin in order to move to documents to the new branch

Change 10146 on 2001/11/16 by jhoule@MA_JHOULE

baseAddr has mipID and gBaseTexOffset

Change 10122 on 2001/11/16 by bbloemer@ma-jasonh

From Michael Litt's far east trip

Change 10117 on 2001/11/16 by bbloemer@ma-jasonh

Updated DDR II spec

Change 10055 on 2001/11/15 by jhoule@MA_JHOULE

Contains instruction filed (both usage in commands, and meanings) as well as constant store fields.

Change 9981 on 2001/11/15 by mdoggett@MA_MDOGGETT

TC diagrams

Change 9979 on 2001/11/15 by mdoggett@MA_MDOGGETT

major rearrangement and integration of old texture pipe material.

Change 9967 on 2001/11/15 by mdoggett@MA_MDOGGETT_LT

no changes

Change 9964 on 2001/11/14 by jhoule@MA_JHOULE

Extensive clock management numbers.

FIFO stalling implemented.

L2Filter implemented using deque for FIFOs of requests at each cacheline.

Created XYBAddr for easier management (the L1 miss sends the exact same texel request to the L2).

Uses a static L2Cache to get correct tags, as well as L2Cache numbers.

Change 9963 on 2001/11/14 by jhoule@MA_JHOULE

Boolean b is now a signed char (%uc was failing).

faf supported in cache config file.

Introduction of baseAddr for read commands (same as mipID for now).

Call of clock() moved before end_of_prim because it uses qQuadPixelCount which would otherwise reset to 0.

STATIC_L2 support.

Change 9962 on 2001/11/14 by jhoule@MA_JHOULE

Uses PRINT_DEBUG define instead of #if 0/1.

Change 9961 on 2001/11/14 by jhoule@MA_JHOULE

Disabled USES_TWO_TAGS.

Change 9940 on 2001/11/14 by mdoggett@MA_MDOGGETT

First check-in of new TC spec. Brings together L1 access from TP spec and old TD spec, plus some additional diagrams and format information.

Change 9923 on 2001/11/14 by jhoule@MA_JHOULE

Changed Wrapping/Clamping to Clamping/Wrapping for subblock naming issues.
Now at 0.8.9.

Change 9915 on 2001/11/14 by jhoule@MA_JHOULE

Blending scheme diagrams.
Modifs here and there.

Change 9910 on 2001/11/14 by dglen@dglen_r400_dell

Moved from arch/doc to devel/doc_lib/parts

Change 9908 on 2001/11/14 by jowang@jowang_R400_win

add 1536 line size so that 1280 doesn't use 4 tap

Change 9886 on 2001/11/14 by dglen@dglen_r400_dell

Moved to doc_lib/parts/dc

Change 9885 on 2001/11/14 by dglen@dglen_r400_dell

Moving files to doc_lib/parts from arch/doc

Change 9859 on 2001/11/14 by lseiler@ma_lseiler

Computes % per bin for HiZ deltas

Change 9850 on 2001/11/14 by llefebvr@llefebvre_laptop_r400

screendiv simulator for HZ precision

Change 9830 on 2001/11/13 by bbloemer@ma-jasonh

Materials from M. Litt's trip to the far east.

Change 9826 on 2001/11/13 by jhoule@MA_JHOULE

Put Sampling in a separate section than Walking, because it is closer to Wrapping.
Safety check-in.

Change 9759 on 2001/11/13 by lseiler@ma_lseiler

numbers library, fixed to eliminate warning messages

Change 9739 on 2001/11/12 by paulv@MA_PVELLA

Initial release.

Change 9738 on 2001/11/12 by paulv@MA_PVELLA

Fixed any grammatical errors found and update of Read Bus Switch, including completion (?) of Tag Sequencer subsection.

Change 9737 on 2001/11/12 by paulv@MA_PVELLA

Updated with latest architectural changes.

Change 9723 on 2001/11/12 by askende@andi_r400_docs

new revision

Change 9715 on 2001/11/12 by lseiler@ma_lseiler

program to test tiling with heirarchical data

Change 9708 on 2001/11/12 by dglen@dglen_r400_dell

Template for display related interface bus specs

Change 9707 on 2001/11/12 by dglen@dglen_r400_dell

Should be ready for the masses.

Change 9653 on 2001/11/09 by jhoule@MA_JHOULE

Week end check-in.
Almost finished stripping out the TC stuff.

Change 9648 on 2001/11/09 by jhoule@MA_JHOULE

Caching is dotted line to indicate sharing between the 4 TPs.

Change 9627 on 2001/11/09 by jhoule@MA_JHOULE

Isolated TC.
Getting ready to strip it out to be plugged in TC (which Michael will check-in afterwards).

Change 9586 on 2001/11/08 by dclifton@dclifton_r400

new specs

Change 9513 on 2001/11/08 by jhoule@MA_JHOULE

Started implementing FIFO stalls with clock notion.

Change 9512 on 2001/11/08 by jhoule@MA_JHOULE

Assert replaced with runtime error message (was crashing because PrimData was NULL).

Change 9511 on 2001/11/08 by jhoule@MA_JHOULE

Cleaner output

Change 9510 on 2001/11/08 by jhoule@MA_JHOULE

QUAKE_HACK (similar to R200_PARSER) to keep rendering within scissor zone.

More robust cache flushing.

Better TAG bits computation (total of 12).

Change 9396 on 2001/11/07 by kcorrell@KCORRELL

continuation of major update working towards review mid Nov

Change 9393 on 2001/11/07 by kcorrell@KCORRELL

updated diagram

Change 9346 on 2001/11/06 by llefebvr@llefebvre_laptop_r400

sequencer spec backup

Change 9255 on 2001/11/06 by jhoule@MA_JHOULE

Single boolean flip hack scheme.

Change 9254 on 2001/11/06 by jhoule@MA_JHOULE

Invalid tag detection for better flushing of cache.

Change 9206 on 2001/11/05 by jhoule@MA_JHOULE

Implement clock(), with FIFO accounting.

Better flip hack scheme (more robust, and simpler to specify).

Change 9204 on 2001/11/05 by jhoule@MA_JHOULE

Support 5 params fa and fa2 (for 2x1 and 2x2 hack of delta 1).

Calls clock().

Change 9199 on 2001/11/05 by jhoule@MA_JHOULE

Added second tag

Change 9198 on 2001/11/05 by jhoule@MA_JHOULE

Added clock() function.

Change 9197 on 2001/11/05 by jhoule@MA_JHOULE

Added clock() function.

Change 9194 on 2001/11/05 by dwong@cndwong2

first draft of R400_IDCT

Change 9118 on 2001/11/02 by jhoule@MA_JHOULE

Reactivated ParseArgs path.

Cleaner support for R200 parser (#defined); also considers TP0 bug workaround from driver!

Wrong pkt deletion solved (crashed after last frame read).

Function isOneTexelTri() added to remove weird (and cache helping) 1-texel triangles (same texCoord at 3 vertices).

RENDER_PRINT_POLY added.

Changed output of 0-textures tris from ! to @.

Change 9116 on 2001/11/02 by jhoule@MA_JHOULE

Removed getchar() when verbosing.

Change 9115 on 2001/11/02 by jhoule@MA_JHOULE

CACHE_VERBOSE came back to 0... (!)

Change 9105 on 2001/11/02 by ctaylor@fl_ctaylor

First Rev of PA Top Level Spec,
VGT_Spec
CLIP/VTE Spec.

Change 9046 on 2001/11/02 by bbloemer@ma-jasonh

AGP 8x spec.

Change 8961 on 2001/11/01 by wlawless@wlawless

.

Change 8932 on 2001/10/31 by kcorrell@KCORRELL

oops this is what was supposed to be submitted the last time...

Change 8927 on 2001/10/31 by kcorrell@KCORRELL

adding some new block diagrams and did a major reorg of the MH document

Change 8718 on 2001/10/29 by jhoule@MA_JHOULE

Added HACK_2x1 for 2x1 cachelines that are effectively EE...

Change 8717 on 2001/10/29 by jhoule@MA_JHOULE

Moved CACHE_VERBOSE to AbsCache for global control.

Change 8716 on 2001/10/29 by jhoule@MA_JHOULE

Aesthetic (output, default caches, etc.)

Change 8687 on 2001/10/29 by jhoule@MA_JHOULE

Uses R200Parser.
Various debug functions and calls.

Change 8675 on 2001/10/29 by jhoule@MA_JHOULE

Uses diags/PM4Tools ParserLib file.

Change 8674 on 2001/10/29 by jhoule@MA_JHOULE

Changed library for diags/PM4Tools one (requires PM4_TOOLS environment variable).
RTTI for CacheSim.cpp only.

Change 8611 on 2001/10/26 by dglen@dglen_r400_dell

Added MH interface detail

Change 8606 on 2001/10/26 by llefebvr@llefebvre_laptop_r400

Sequencer spec V1.0.

Change 8413 on 2001/10/24 by jowang@jowang_R400_win

1) added some new scaling situations

2) made max VTAP equal to 4 in in 30BPP

Change 8404 on 2001/10/24 by mdoggett@MA_MDOGGETT

Small interface corrections and additions.

Change 8386 on 2001/10/24 by kcorrell@KCORRELL

block diagram of MHC and MHA

Change 8377 on 2001/10/24 by jowang@jowang_R400_win

just testing

Change 8376 on 2001/10/24 by jowang@jowang_R400_win

just testing

Change 8371 on 2001/10/24 by jowang@jowang_R400_win

no change

Change 8358 on 2001/10/24 by lseiler@ma_lseiler

Render Backend Registers v0.2: Various changes from one-on-one reviews

Change 8326 on 2001/10/24 by jhoule@MA_JHOULE

Added $(R400_EMU) to include paths

Change 8325 on 2001/10/24 by jhoule@MA_JHOULE

Key 'K' uses emulator wrapper function.

Key '@' get monochromatic mipmaps.

Change 8324 on 2001/10/24 by jhoule@MA_JHOULE

6 bits for mipW

mipmap nearest LOD bias reactived

3 bits LOD correction factors

getLODUsingEmu wrapper for emulator comparisons

Change 8321 on 2001/10/23 by dglen@dglen_r400_dell

Add more detail to interlaced display modes.

Change 8319 on 2001/10/23 by dglen@dglen_r400_dell

Top level architectural outline of new display system.

Change 8256 on 2001/10/23 by paulv@MA_PVELLA

Updated DBR and RRA sections, with updates to functional and architectural descriptions, along with updates to the block diagrams. Also started Tag Sequencer subsection.

Change 8211 on 2001/10/22 by dglen@dglen_r400_dell

Updated revision history

Change 8209 on 2001/10/22 by dglen@dglen_r400_dell

Top level outline of the front end VGA core unit.

Change 8207 on 2001/10/22 by dglen@dglen_r400_dell

Template for Display Block Architectural Specs

Change 8206 on 2001/10/22 by dglen@dglen_r400_dell

Display output image scaler modes and BW requirements

Change 8175 on 2001/10/22 by llefebvr@llefebvre_laptop_r400

sequencer v1.0 BACKUP ONLY not yet complete.

Change 8167 on 2001/10/22 by paulv@MA_PVELLA

Forgot to fix the DISP routing. It is done.

Change 8161 on 2001/10/22 by wlawless@wlawless

new file

Change 8152 on 2001/10/22 by beiwang@MA_BEIWANG

Power management related links

Change 8148 on 2001/10/22 by rbeaudin@MA_RAYB_LT

moved numbers stuff

Change 8146 on 2001/10/22 by paulv@MA_PVELLA

Fixed block diagram to reflect recent functional/architectural changes.

Change 8114 on 2001/10/19 by kcorrell@KCORRELL

Modified connections to AIC interface block (HMB)

Change 8108 on 2001/10/19 by askende@andi_r400_docs

deleted ..it was used as a test only

Change 8106 on 2001/10/19 by jayw@MA_JAYW

Jay's exploration of recip

Change 8105 on 2001/10/19 by askende@andi_r400_docs

test

Change 8081 on 2001/10/19 by llefebvr@llefebvre_laptop_r400

One before last major architectural revision of the sequencer before the implementation spec. Control flow is complete and was accepted by SW team. Remains before freezing 1.0 : external and internal interfaces.

Change 8039 on 2001/10/18 by lseiler@ma_lseiler

Moved sized int definitions into standard_typedefs.h

Change 8011 on 2001/10/18 by mdoggett@MA_MDOGGETT

a few interface changes

Change 7974 on 2001/10/17 by paulv@MA_PVELLA

Refixed block diagram (what I had before last revision was mostly correct).

Change 7934 on 2001/10/17 by llefebvr@llefebvre_laptop_r400

diagrams that go with the sequencer spec

Change 7930 on 2001/10/17 by llefebvr@llefebvre_laptop_r400

version 0.8 of the sequencer spec. It contains the new control flow porcedure as well as updated external interfaces.

Change 7923 on 2001/10/17 by jhoule@MA_JHOULE

Removed PRINT field.

Change 7884 on 2001/10/16 by jhoule@MA_JHOULE

Added constant store and instruction store in the interface portion.

Change 7850 on 2001/10/16 by lseiler@SEILER2

Changed int8 etc. to #define constants protected by #ifndef
This avoids an error if they are defined in another file

Change 7824 on 2001/10/16 by jhoule@MA_JHOULE

Cleaning up of interfaces. Removed duplicates.

Change 7809 on 2001/10/15 by paulv@MA_PVELLA

Fixed DBR and RRA sections with new I/O and logic adjustments.

Change 7808 on 2001/10/15 by paulv@MA_PVELLA

Updated block diagram to fix I/O and some logic changes.

Change 7806 on 2001/10/15 by paulv@MA_PVELLA

Updated block diagram to include AIC address and byte swap I/O.

Change 7779 on 2001/10/15 by jhoule@MA_JHOULE

Minor corrections here and there.

Change 7722 on 2001/10/12 by paulv@MA_PVELLA

Minor fix the the RRA block diagram (changed the tag info FIFO to a tag info RAM).

Change 7717 on 2001/10/12 by mdoggett@MA_MDOGGETT

First version of a document describing how shadow map calculations can be performed on the R400 using plane equations.

Change 7705 on 2001/10/12 by rbagley@MA_RBAGLEY_LT

First version of investigation into worst-case consequences of software features and hardware attributes for shader programs.

Change 7684 on 2001/10/12 by jhoule@MA_JHOULE

Major reorganization of sections.
Not revision-savvy.

Change 7683 on 2001/10/12 by jhoule@MA_JHOULE

Added Output Formatter at the end (regroups into a quad, and converts to floats).

Change 7645 on 2001/10/11 by paulv@MA_PVELLA

Fixed Data Bus Router (HDP client needed to be added).

Change 7631 on 2001/10/11 by jhoule@MA_JHOULE

New Block Description (reorganization underway).

Change 7616 on 2001/10/11 by bbloemer@ma-jasonh

Power management

Change 7615 on 2001/10/11 by paulv@MA_PVELLA

Minor fix.

Change 7611 on 2001/10/11 by lseiler@ma_lseiler

First draft of Render Backend register field list

Change 7581 on 2001/10/11 by paulv@MA_PVELLA

Updated block diagram to reflect recent changes (see spec).

Change 7580 on 2001/10/11 by paulv@MA_PVELLA

Updated Data Bus Router section.

Change 7572 on 2001/10/11 by bbloemer@ma-jasonh

New material

Change 7553 on 2001/10/10 by bbloemer@ma-jasonh

Power management spec.

Change 7487 on 2001/10/09 by jhoule@MA_JHOULE

Accepted all changes.
Various minor tweaks here and there (notably, issues aren't headings anymore).

Change 7461 on 2001/10/09 by jhoule@MA_JHOULE

Better add the top-level diagram if I want people to see it...

Change 7460 on 2001/10/09 by jhoule@MA_JHOULE

Added new top-level diagram.
Description of pipe in features to give a good idea of where we are heading.

Change 7418 on 2001/10/05 by paulv@MA_PVELLA

Another big update to RRA (with new understanding of subblock, functionality defined in much better detail). Also fixed DBR subblock section to reflect removal of BUFFER_FULL flags (now done in RRA).

Change 7417 on 2001/10/05 by paulv@MA_PVELLA

Updated diagram to reflect recent changes/adjustments.

Change 7416 on 2001/10/05 by paulv@MA_PVELLA

No longer valid. Replaced by MH-RRA.vsd.

Change 7411 on 2001/10/05 by jhoule@MA_JHOULE

Week end check in (minor corrections, mostly formatting).

Change 7380 on 2001/10/05 by llefebvr@llefebvre_laptop_r400

version 0.7 of the sequencer. Interfaces and control managment added.

Change 7371 on 2001/10/05 by mdoggett@MA_MDOGGETT

small changes to MC interfaces

Change 7370 on 2001/10/05 by mdoggett@MA_MDOGGETT

0.4 revision, final check in for major changes
(that's the idea at least)

Change 7361 on 2001/10/05 by jhoule@MA_JHOULE

Walkers more up-to-date.
Described a bit the samples fetching (pt, bilin, arbi).

Change 7350 on 2001/10/04 by paulv@MA_PVELLA

New block diagram for Read Return Arbitrator (RRA).

Change 7349 on 2001/10/04 by paulv@MA_PVELLA

Added block diagram to RRA section along with more documentation (functional/interface).

Change 7335 on 2001/10/04 by jhoule@MA_JHOULE

Updated the LOD computation pseudo-code.
This has to update walkers (with diagram), and the clamp logic (since it's now in float around [0,1] with clamp and mirror flags).

Change 7311 on 2001/10/04 by bbloemer@ma-jasonh

Early rev of spec.

Change 7261 on 2001/10/03 by llefebvr@llefebvre_laptop_r400

backup of the sequencer + register loading diagram

Change 7192 on 2001/10/02 by paulv@MA_PVELLA

Some minor updates.

Change 7149 on 2001/10/01 by wlawless@wlawless

nothing

Change 7113 on 2001/10/01 by bbloemer@ma-jasonh

Another Elpida spec

Change 7102 on 2001/09/28 by paulv@MA_PVELLA

Fixed the data bus router's block diagram. Some minor document tweaks.

Change 7088 on 2001/09/28 by mdoggett@MA_MDOGGETT

More changes.

Change 7080 on 2001/09/28 by llefebvr@llefebvre_laptop_r400

New sequencer diagram

Change 7050 on 2001/09/27 by paulv@MA_PVELLA

A portion of the old Read Return Arbiter has become what is now known as the Data Bus Router. This is the block diagram.

Change 7044 on 2001/09/27 by jhoule@MA_JHOULE

Aesthetic modifications.
Table of contents now has correct number sizes!!! (hurray!)
Thanks to Mike "Vege-mate" Doggett for this...

Change 7027 on 2001/09/27 by bbloemer@ma-jasonh

Added Chaplin RAM

Change 7008 on 2001/09/27 by paulv@MA_PVELLA

This is the initial checkin of the read return abitrator block diagram.

Change 6931 on 2001/09/26 by lseiler@SEILER2

This tests printing uint8 and int8

Change 6889 on 2001/09/25 by askende@andi_docs

newest version

Change 6878 on 2001/09/25 by mdoggett@MA_MDOGGETT

major changes to document structure and content.
progress checkin before version 0.4

Change 6865 on 2001/09/24 by llefebvr@llefebvre_laptop_r400

new spec of the Sequencer.

Change 6843 on 2001/09/24 by lseiler@ma_lseiler

This version eliminates most templating on the tiling.h classes and eliminates the more complex caching classes. This is more efficient and makes the code easier to understand. Now that we are fixed on 8x8 tiles and a cache with a small number of sets (maybe just 1), we don't need the extra complexity.

Change 6833 on 2001/09/24 by wlawless@wlawless

updated revision code

Change 6790 on 2001/09/21 by llefebvr@llefebvre_laptop_r400

RE spec backup + HZ stats + SC spec backup

Change 6789 on 2001/09/21 by llefebvr@llefebvre_laptop_r400

new diagrams for the sequencer

Change 6758 on 2001/09/20 by wlawless@wlawless

Slight update

Change 6722 on 2001/09/20 by bbloemer@ma-jasonh

For tonight's call.

Change 6561 on 2001/09/17 by llefebvr@llefebvre_laptop_r400

more statistical data on HZ

Change 6526 on 2001/09/14 by lseiler@SEILER2

Added int8 and uint8 ostream operation: now outputs number, not char
Added andField, orField, and xorField functions (old names: and, or, xor)
Added bits and mode fields to floor, trunc, round, ceil for fixed_hw types (were already defined for the software types)

Change 6317 on 2001/09/11 by jhoule@MA_JHOULE

Minor LODBias correction in LOD computation pseudo-code.

Change 6309 on 2001/09/11 by rbagley@MA_RBAGLEY

Several emendations.

Change 6243 on 2001/09/10 by lseiler@ma_lseiler

Changed to 8x8 tiling to compute EmptyQuad slots for render backend

Change 6200 on 2001/09/07 by lseiler@ma_lseiler

Tiling: added code to compute extra quads processed by the shader pipe due to RB crossbar swizzling

Change 6184 on 2001/09/07 by lseiler@SEILER2

Update to fix compatibility bug with gcc v3.0

Change 6183 on 2001/09/07 by lseiler@SEILER2

Fixes a compatiability problem with gcc v3.0

Change 5994 on 2001/08/31 by llefebvr@llefebvre_laptop_r400

removed dead code

Change 5970 on 2001/08/30 by bbloemer@ma-jasonh

Infineon updates.

Change 5872 on 2001/08/29 by jhoule@MA_JHOULE

Frequency dumping code added, different clamping scheme.

Change 5871 on 2001/08/29 by jhoule@MA_JHOULE

Camera dumping added.

Change 5786 on 2001/08/28 by sallen@devel_sallen

fix compile errors .....

Change 5777 on 2001/08/28 by lseiler@ma_lseiler

Version 0.5a -- fixed typos in the bus interface tables

Change 5776 on 2001/08/28 by jhoule@MA_JHOULE

Various crap (safety check-in)

Change 5720 on 2001/08/24 by bbloemer@ma-jasonh

300 MHz. Chaplain class parts.

Change 5711 on 2001/08/24 by llefebvr@llefebvre_laptop_r400

new version of the registry file allocation mechanism

Change 5702 on 2001/08/24 by bbloemer@ma-jasonh

8/23/01 Samsung Documents

Change 5698 on 2001/08/24 by llefebvr@llefebvre_laptop_r400

version 0.4 of the sequencer

Change 5537 on 2001/08/21 by wlawless@wlawless

RB Hardware Design Spec

Change 5466 on 2001/08/17 by askende@andi_docs

new rev of the spec

Change 5465 on 2001/08/17 by jhoule@MA_JHOULE

Better LOD algorithm (leaner and better... now uses log2)

Change 5456 on 2001/08/17 by jhoule@MA_JHOULE

Converted to log2 everywhere for AnisoNew

Change 5448 on 2001/08/16 by rbagley@MA_RBAGLEY_LT

Bit formats for preliminary fetch instructions, discussion and
handling of defaults, MOV macros, and miscellaneous corrections.

Change 5433 on 2001/08/16 by jhoule@MA_JHOULE

R400AnisoNew modified to represent current state of spec.

Change 5387 on 2001/08/15 by jhoule@MA_JHOULE

Changed R400AnisoNew to an odd number of samples, with half-weighted endpoints (pretty good looking!!! still got the LOD transition to fix).

Change 5384 on 2001/08/15 by jhoule@MA_JHOULE

Initial check-in.
This file gives supplemental information on design decisions, notations, algorithms and various other underlying stuff.
It helps explain more, but reduce the already too big texture spec.

Change 5383 on 2001/08/15 by jhoule@MA_JHOULE

File wasn't saved (sorry).

Change 5382 on 2001/08/15 by jhoule@MA_JHOULE

Added LOD computation (with correction), samples walking (trilinear and anistropic), and texel fetching (with wrapping policy table) in the logic description.

Change 5313 on 2001/08/14 by jhoule@MA_JHOULE

New R200 scheme (hopefully more closely matching).

Aslo #if...#endif correction factors printing.

Change 5289 on 2001/08/13 by llefebvr@llefebvre_laptop_r400

added an exemple of registry file management

Change 5278 on 2001/08/13 by llefebvr@llefebvre_laptop_r400

A more optimized version without while loops

Change 5260 on 2001/08/13 by llefebvr@llefebvre_laptop_r400

updated spec for sequencer

Change 5259 on 2001/08/13 by llefebvr@llefebvre_laptop_r400

Reference version for registry file management. Working but costly.

Change 5240 on 2001/08/10 by llefebvr@llefebvre_laptop_r400

project was missing some files

Change 5239 on 2001/08/10 by llefebvr@llefebvre_laptop_r400

working visual simulation of the dynamic allocation of the register file

Change 5197 on 2001/08/09 by rbagley@MA_RBAGLEY

Updates to scalar alu and fetch operations, and miscellaneous correction.

Change 5175 on 2001/08/09 by lseiler@ma_lseiler

Memory Controller, v0.5: removed autotag feature, changed bus/signal names and other stuff

Change 5163 on 2001/08/09 by jhoule@MA_JHOULE

Missed the author field. Fixed also.

Change 5161 on 2001/08/09 by jhoule@MA_JHOULE

Put title in a title field.

Change 5159 on 2001/08/09 by jhoule@MA_JHOULE

Changed the year from 2000 to 2001.

Change 5055 on 2001/08/07 by jhoule@MA_JHOULE

getLODR{23}00Corrected disappeared (wrapped inside the getLODR{23}00.

Added approxLength for sqrt(a,b) approximation.

Change 5054 on 2001/08/07 by jhoule@MA_JHOULE

Different LODfunc scheme (now has independent control over LOD correction with '-')

Change 5053 on 2001/08/07 by mdoggett@MA_MDOGGETT

Updated table of contents

Change 5052 on 2001/08/07 by mdoggett@MA_MDOGGETT

Added new visio diagrams. Updates to most sections, particularly Logic Descriptions.

Change 5046 on 2001/08/06 by jhoule@MA_JHOULE

Added #if 1...#endif for selecting 1 jitter/pixel or 1jitter/quad

Change 5045 on 2001/08/06 by llefebvr@llefebvre_laptop_r400

multiple jiitering patterns for the simulator

Change 5036 on 2001/08/03 by jhoule@MA_JHOULE

Cleanup of centroid-related functions.

Change 5032 on 2001/08/03 by jhoule@MA_JHOULE

New centroids support (jittered pattern).

New anisotropies also.

Bool flag for when anisotropy is 1x1 (r200 aniso can use trilinear).

Change 5031 on 2001/08/03 by jhoule@MA_JHOULE

RGBA in QuadData now is float (new scheme will support any type).

New anistropies supported.

Change 5029 on 2001/08/03 by jhoule@MA_JHOULE

New sampling patterns implemented.

Change 5028 on 2001/08/03 by jhoule@MA_JHOULE

Aesthetic

Change 5027 on 2001/08/03 by llefebvr@llefebvre_laptop_r400

new rasteriser simulator using a real jitter table

Change 5016 on 2001/08/03 by llefebvr@llefebvre_laptop_r400

setup unit revised spec

Change 5015 on 2001/08/02 by rbagley@MA_RBAGLEY

Clarfication of texture fetch instruction.

Change 4989 on 2001/08/01 by rbagley@MA_RBAGLEY

First version of texture instructions and some further emendations.

Change 4987 on 2001/08/01 by jhoule@MA_JHOULE

Seems to have solved R200 anisotropy (check for trilinear artifacts).

LOD shows in R400.

Change 4980 on 2001/08/01 by llefebvr@llefebvre_laptop_r400

new spec for SC and RE. Changed a bit the interface to encompass the fact that RE is now using the full precision normalized slopes from the SU because it takes the same number of bits than doing a per tile compression for the barycentric coordinates and it also simplifies a lot the SC.

Change 4960 on 2001/08/01 by askende@andi_docs

new rev

Change 4955 on 2001/08/01 by jhoule@MA_JHOULE

Changed R200 aniso (and it now supports trilinear when aniso == 1).

Solved bug (fabs was too early, and lost when sign was different).

Added setColor.

Change 4929 on 2001/07/31 by askende@andi_docs

a new rev

Change 4922 on 2001/07/31 by lseiler@SEILER2

Added support for floating point and group exponent ranges smaller than [0..1)
Added an optional parameter to round, ceil, etc. that specicies how many low order bits of destination precision to ignore in the conversion (defaults to zero)

Change 4897 on 2001/07/31 by bbloemer@ma-jasonh

Non-confidential description of IML technology.

Change 4866 on 2001/07/30 by jhoule@MA_JHOULE

Trilinear anisotropy (R400 style).

Change 4854 on 2001/07/30 by mdoggett@MA_MDOGGETT

Updates to most sections.

Change 4849 on 2001/07/30 by lseiler@SEILER2

Added code to find real MIN and MAX for class number::

Change 4816 on 2001/07/27 by jhoule@MA_JHOULE

Added changeLODFunc(delta)

Cleaner switch with anisotropy and others.

Change 4811 on 2001/07/27 by jhoule@MA_JHOULE

Added anisotropy support.

'k' resets LOD technique (because aniso breaks it).

Change 4810 on 2001/07/27 by jhoule@MA_JHOULE

Working R200 anisotropy implemented.

Added various functions, code is not cleaned up.

Change 4779 on 2001/07/26 by rbagley@MA_RBAGLEY

Corrections and additions to the export specification.

Change 4736 on 2001/07/26 by jacarey@fl_jacarey

Moved Original CP and RBBM Specs from ../arch/doc to ../doc_lib/chip/<unit> areas.

Change 4722 on 2001/07/25 by rbagley@MA_RBAGLEY

Emendations to the export specification.

Change 4694 on 2001/07/25 by llefebvr@llefebvre_laptop_r400

new version of Pnm for better sampling the noise

Change 4668 on 2001/07/24 by jhoule@MA_JHOULE

Added @mipmap support ('i').

LOD technique selection uses increment count (easier add-ins).

Change 4666 on 2001/07/24 by lseiler@SEILER2

added debug printouts for disallowed cases

Change 4659 on 2001/07/24 by jhoule@MA_JHOULE

strncpy doesn't access NULL string as source.

Change 4644 on 2001/07/23 by rbagley@MA_RBAGLEY

First version of the export specification and additional corrections.

Change 4640 on 2001/07/23 by jhoule@MA_JHOULE

Added getLODTheo to distinguish artifacts that are from LOD error in opposition to multi-sampling ones.

Change 4639 on 2001/07/23 by jhoule@MA_JHOULE

Better default directory support (I think this is how it works... anyways, it seems to work OK).

Change 4629 on 2001/07/23 by lseiler@SEILER2

Changed name from Test.h to test.h

Change 4628 on 2001/07/23 by lseiler@SEILER2

Deleting to change name to lower case

Change 4627 on 2001/07/23 by lseiler@SEILER2

About to change the name to lower case

Change 4619 on 2001/07/23 by lseiler@SEILER2

Minor bug fix for comparing integer classes

Change 4598 on 2001/07/20 by jhoule@MA_JHOULE

Can now decrement LOD technique (2-way).

Change 4597 on 2001/07/20 by jhoule@MA_JHOULE

Many minor corrections: some aesthetic, some (hopefully) helpful to understand more clearly, others are comments for Steve.

Change 4593 on 2001/07/20 by lseiler@SEILER2

Fixed rounding/truncation etc., eliminated minor bugs and warning messages

Change 4586 on 2001/07/20 by jhoule@MA_JHOULE

Minor changes: changed header for cleaner ones as well as Title Field (no more Top Level Spec).

Change 4584 on 2001/07/20 by jhoule@MA_JHOULE

Integrated LOD correction.

Default path for scenes and textures.

Quad and Dome creation (for Scn files).

Added AA support (walker variables set for texture).

Print(QuadData) now shows bits for aamask.

Change 4581 on 2001/07/20 by jhoule@MA_JHOULE

LOD correction mostly solved (requires simple H & V factors to account for width & height of pixel with samples).

Added LOD functions (w or w/o correction) as well as correction functions.

Change 4542 on 2001/07/19 by lseiler@SEILER2

Supports group exponent numbers

Change 4537 on 2001/07/19 by mdoggett@MA_MDOGGETT

Fixed header to not have different odd and even headers.

Change 4517 on 2001/07/18 by jhoule@MA_JHOULE

Temp check-in (for old LOD correction)

Change 4431 on 2001/07/17 by lseiler@ma_lseiler

(shouldn't have been checked in)

Change 4425 on 2001/07/17 by lseiler@ma_lseiler

automatically generated files that don't need to be checked in

Change 4424 on 2001/07/17 by lseiler@ma_lseiler

DirectX shared files

Change 4423 on 2001/07/17 by lseiler@ma_lseiler

Animation test for multisampling (from R300 team)

Change 4420 on 2001/07/17 by jhoule@MA_JHOULE

Added makeQuad for AA testing.

Change 4400 on 2001/07/16 by jhoule@MA_JHOULE

Anti-aliasing support added.

Change 4399 on 2001/07/16 by jhoule@MA_JHOULE

Added clamped and convert functions.

Change 4398 on 2001/07/16 by jhoule@MA_JHOULE

Calls renderAA and endRender at appropriate places.

Removed old makeMovie trace code.

Change 4397 on 2001/07/16 by jhoule@MA_JHOULE

Anti-aliasing support added.

Change 4384 on 2001/07/16 by jhoule@MA_JHOULE

Starting AA buffer (recoded the render to iterate from 0 to 3)

Change 4366 on 2001/07/16 by jhoule@MA_JHOULE

Depth and color types can be changed.

Change 4322 on 2001/07/13 by jhoule@MA_JHOULE

Wrongly associated w&h with t&s in getLODRx00 (didn't show because of square textures).

(some traces must be removed)

Change 4310 on 2001/07/13 by jhoule@MA_JHOULE

Last mipmap level is now gray instead of white.

Change 4258 on 2001/07/12 by jhoule@MA_JHOULE

Mipmap dumping with 'd' key.

Added area LOD computation.

Change 4230 on 2001/07/11 by rbagley@MA_RBAGLEY

A skein of corrections and minor changes have been made. The
version of the document remains 1.0.

Change 4205 on 2001/07/11 by lseiler@ma_lseiler

More v0.2 files

Change 4203 on 2001/07/11 by lseiler@ma_lseiler

These files are outmoded

Change 4201 on 2001/07/11 by lseiler@ma_lseiler

These files are outmoded

Change 4200 on 2001/07/11 by lseiler@ma_lseiler

Render Backend: version 0.2 with greatly revised Paramater Buffer and Tile Logic sections

Change 4179 on 2001/07/11 by jhoule@MA_JHOULE

Consistent default filter values between OpenGL and SW renderer.

Change 4178 on 2001/07/11 by jhoule@MA_JHOULE

Color mipmap now has white as end colors.

Change 4173 on 2001/07/10 by rbagley@MA_RBAGLEY

Add first version of a syntax specification for the R400 shader
programming language.

Change 4033 on 2001/07/06 by askende@andi_docs

update of the specs

Change 4020 on 2001/07/06 by jhoule@MA_JHOULE

RELEASE_04_June/20/01

1. VB reconstruction optimized. Indx_Buffer packet implemented but currently not working properly.
2. Pm4stats finalized and now working properly on pixel counting
3. Dword-aligned indx_buffer packet support
4. Modified code to avoid WatCom compiling warnings. (Though some lines look stupid).

Change 4019 on 2001/07/06 by jhoule@MA_JHOULE

New version coming.

Change 4015 on 2001/07/06 by jhoule@MA_JHOULE

Changed RS_TRACE to LS_TRACE

Change 4010 on 2001/07/06 by smorein@smorein_r400

major texture pipe spec update- complete except for addresing logic.
Could use some more editing, and probably more block diagrams.

Change 4006 on 2001/07/05 by jhoule@MA_JHOULE

Fast OpenGL refresh with @ sign.

Change 4001 on 2001/07/05 by llefebvr@llefebvre_laptop_r400

lockin is on

Change 4000 on 2001/07/05 by llefebvr@llefebvre_laptop_r400

sequencer checkin

Change 3999 on 2001/07/05 by pmitchel@pmitchel_iris

change file type to lock

Change 3996 on 2001/07/05 by pmitchel@pmitchel_iris

change filetype to prevent simultaneous open for edit

Change 3995 on 2001/07/05 by pmitchel@pmitchel_iris

change file type to prevent simultaneous open for edit

Change 3994 on 2001/07/05 by llefebvr@llefebvre_laptop_r400

updated scan converter spec

Change 3993 on 2001/07/05 by jhoule@MA_JHOULE

Support for negative or 0 loops.

Change 3991 on 2001/07/05 by lseiler@ma_lseiler

Tiling: deletes and recreates the RasterMain class between files -- hopefully this will keep the virtual memory required from increasing over multi-file runs

Change 3990 on 2001/07/05 by jhoule@MA_JHOULE

Better screenshot support (more intelligent prompting)

Change 3987 on 2001/07/05 by pmitchel@pmitchel_iris

changed filetype to strict locking

Change 3982 on 2001/07/05 by jhoule@MA_JHOULE

Pretty versatile animation support (all camera).

Added loop support, as well as screen refresh (for previews).

Change 3976 on 2001/07/05 by lseiler@ma_lseiler

Tiling Code: defining MEM_OUT outputs memory access files

Change 3908 on 2001/07/03 by lseiler@ma_lseiler

FrameBuf v0.3: some extra compressed and 3d formats, minor changes

Change 3905 on 2001/07/03 by jhoule@MA_JHOULE

Slightly better sampling (thanks to MD)

Change 3901 on 2001/07/03 by jhoule@MA_JHOULE

More robust parsing.

Can now save a snapshot of ColorBuffer.

Change 3897 on 2001/07/03 by jhoule@MA_JHOULE

Cleaner code (using macro)

Change 3896 on 2001/07/03 by jhoule@MA_JHOULE

Added and integrated superFilter, which makes multiple samples for every pixel.

Change 3888 on 2001/07/03 by jhoule@MA_JHOULE

Removed old commented trilinear code (now works with muliPass2)

Change 3885 on 2001/07/03 by jhoule@MA_JHOULE

loadScene supports recursive loads (with loadScene "filename")

Change 3884 on 2001/07/02 by jhoule@MA_JHOULE

Added loadTexture in scene file.

Change 3882 on 2001/07/02 by jhoule@MA_JHOULE

Removed forcePerspective.

Added loadScene with interface hooks.

Change 3879 on 2001/07/02 by jhoule@MA_JHOULE

Aesthetic

Change 3878 on 2001/07/02 by jhoule@MA_JHOULE

Added toGL and createColorMimap (each level is a different color).

Change 3877 on 2001/07/02 by jhoule@MA_JHOULE

Trilinear fully functional.

Change 3871 on 2001/07/02 by mdoggett@MA_MDOGGETT

Minor updates to Memory Hub spec

Change 3870 on 2001/07/02 by jhoule@MA_JHOULE

Working filter (min/mag with nearest/{bi}linear and mipmaps!)

Change 3867 on 2001/07/02 by jhoule@MA_JHOULE

Initial check-in.

Change 3866 on 2001/07/02 by jhoule@MA_JHOULE

Removed old filter code.

Change 3842 on 2001/06/29 by jhoule@MA_JHOULE

GlutMouseEvent added (with other stuff)

Change 3841 on 2001/06/29 by jhoule@MA_JHOULE

Camera is integrated (with a cool bounding box for OpenGL depths)

Change 3838 on 2001/06/29 by jhoule@MA_JHOULE

Towards camera integration.

Change 3834 on 2001/06/29 by jhoule@MA_JHOULE

Added numLevels and getLevel(int)

Change 3833 on 2001/06/29 by jhoule@MA_JHOULE

Removed trace

Change 3832 on 2001/06/29 by jhoule@MA_JHOULE

Added bilinearFetch.

Distinction between TexColor (copy, read-write) and TexColorP (pointer, read-only)

Change 3830 on 2001/06/29 by jhoule@MA_JHOULE

These files don't need RTTI.

Change 3812 on 2001/06/28 by jhoule@MA_JHOULE

Mipmap files added.

Change 3811 on 2001/06/28 by jhoule@MA_JHOULE

Mipmap integration.

Added fetcher (for a per pixel basis). Added TexColor to clean up interface to GLMap<>.

Change 3810 on 2001/06/28 by jhoule@MA_JHOULE

Mipmap integration.

LOD limiting in interface.

Change 3767 on 2001/06/27 by wlawless@wlawless_ws

Updated the stencil area

Change 3764 on 2001/06/27 by bbloemer@ma-jasonh

kj

Change 3755 on 2001/06/27 by jhoule@MA_JHOULE

Missed a 1 for non-square sizes.

Change 3754 on 2001/06/27 by jhoule@MA_JHOULE

Added copyTo and numChannels

Change 3753 on 2001/06/27 by jhoule@MA_JHOULE

Initial CI.

Creation from GLMap done.

Only accepts power of 2 textures.

Non-square downSampling not tested (but should work).

Change 3749 on 2001/06/26 by bbloemer@ma-jasonh

Timing diagram.

Change 3709 on 2001/06/26 by jhoule@MA_JHOULE

Added mirrorOnce support.

Change 3705 on 2001/06/26 by jhoule@MA_JHOULE

Precision-removing code for interpolator weights done by hand.

Change 3697 on 2001/06/25 by beiwang@MA_BEIWANG

test

Change 3691 on 2001/06/25 by wlawless@wlawless_ws

Added a total block

Change 3681 on 2001/06/25 by lseiler@ma_lseiler

Superceded by R400_FrameBuf.vsd

Change 3680 on 2001/06/25 by lseiler@ma_lseiler

Frame Buffer Format, v0.2: complete rewrite

Change 3665 on 2001/06/22 by jhoule@MA_JHOULE

Added numeric library.

Change 3664 on 2001/06/22 by jhoule@MA_JHOULE

Added translateScene and makeMovie.

Uses dumpTIFF instead of dumpRGB.

Change 3663 on 2001/06/22 by jhoule@MA_JHOULE

Week end check-in.

Precision integration with numerical library.

Change 3658 on 2001/06/22 by bbloemer@ma-jasonh

For biweekly call.

Change 3656 on 2001/06/22 by jhoule@MA_JHOULE

dumpRGB is now dumpTIFF (but use it only for RGB{A} GLMaps.

Change 3652 on 2001/06/22 by jhoule@MA_JHOULE

Added title support.

Change 3632 on 2001/06/22 by wlawless@wlawless_ws

Added color and depth compress to RB Size

Change 3614 on 2001/06/21 by jhoule@MA_JHOULE

OpenGL matching of supported WRAP and FILTER

Change 3609 on 2001/06/21 by jhoule@MA_JHOULE

Better printing when necessary.

Change 3608 on 2001/06/21 by jhoule@MA_JHOULE

R200 rasterizer.
Tile size should be 4x4, and even with that, texture coordinates seem to be wrong with magnification cases.
Project dropped (this is if we ever come back to it)

Change 3607 on 2001/06/21 by jhoule@MA_JHOULE

Removed R200 rasterizer.

Change 3606 on 2001/06/21 by jhoule@MA_JHOULE

Added makeDefaultTexture.

Support for R200 rasterizer (to be removed).

Change 3590 on 2001/06/21 by jhoule@MA_JHOULE

RELEASE_03_June/20/01

1. VB reconstruction for Rage200 implemented except indx_buff packet
2. pm4stats finished
3. Fixed bugs in culling function and 3d_draw_indx packet
4. Added register initialization function
5, Added more defensive code warning the unimplemented packets
6. Added version info in client programs. Parser class release date and version showes in usage info.

Change 3589 on 2001/06/21 by jhoule@MA_JHOULE

New version (3) coming in...

Change 3588 on 2001/06/21 by smorein@smorein_r400

Major update. There are still a bunch of inconsistancies (and misspellings) but enough there for people closely associated with the block to need to go through it.

Change 3585 on 2001/06/20 by askende@andi_docs

new rev

Change 3582 on 2001/06/20 by jhoule@MA_JHOULE

FileDialog integration.

Added loadTexture and takeSnapshot.

Different makeScene (more versatile once more).

Uses 'm,./' for wrapping policy.

Change 3581 on 2001/06/20 by jhoule@MA_JHOULE

Added mirror. Solved bug in repeat (had to floor, not just truncate).

Many traces (commented) need to be cleaned up a bit.

Change 3575 on 2001/06/20 by wlawless@wlawless_ws

Added a new RB page with more detail

Change 3574 on 2001/06/20 by askende@andi_docs

new rev

Change 3569 on 2001/06/20 by wlawless@wlawless_ws r400 area

Change 3565 on 2001/06/19 by askende@andi_docs

new rev

Change 3564 on 2001/06/19 by bbloemer@ma-jasonh

500 MHz. presentation

Change 3563 on 2001/06/19 by jhoule@MA_JHOULE

Clamping functions now in class (with pointer-to-method for easy selection at runtime)

Change 3562 on 2001/06/19 by jhoule@MA_JHOULE

Initial check-in.

Change 3561 on 2001/06/19 by jhoule@MA_JHOULE

Little more robust loading.

Change 3560 on 2001/06/19 by askende@andi_docs

new rev

Change 3558 on 2001/06/19 by askende@andi_docs

another rev

Change 3557 on 2001/06/19 by jhoule@MA_JHOULE

Very minor modifs (was a project file problem)

Change 3556 on 2001/06/19 by jhoule@MA_JHOULE

Remade from scratch

Change 3555 on 2001/06/19 by jhoule@MA_JHOULE

Remade the file from scratch

Change 3553 on 2001/06/19 by askende@andi_docs

another rev (rev.03) of the shader spec

Change 3550 on 2001/06/19 by jhoule@MA_JHOULE

mFrameBuffer is now mColorBuffer (dependencies were screwed)

Change 3549 on 2001/06/19 by jhoule@MA_JHOULE

Added refreshBuffer().

Interface can now change filtering in R400 rasterization also.

More versatile scene creation.

Change 3547 on 2001/06/18 by jhoule@MA_JHOULE

Working bilinear with wrapping.

Change 3546 on 2001/06/18 by jhoule@MA_JHOULE

New wrapping, all done in integer (strictly positive).

Bilinear filter roughly coded.

Change 3537 on 2001/06/18 by jhoule@MA_JHOULE

Removed unnecessary wrap.

Change 3536 on 2001/06/18 by jhoule@MA_JHOULE

Added beginText/endText so that we can more easily distinguish OpenGL/R400.

Changed the scene a bit.

Change 3535 on 2001/06/18 by jhoule@MA_JHOULE

Changed default color.

Change 3523 on 2001/06/15 by jhoule@MA_JHOULE

Added TexUnit, and integrated filtering.

Copied texture params & such from other project.

Cool ascii texture (commented).

Change 3522 on 2001/06/15 by jhoule@MA_JHOULE

Added wrap and clamp (needs cleaner interface).

Change 3521 on 2001/06/15 by jhoule@MA_JHOULE

ImageLoader and TexUnit added.

ZBuffer removed.

Change 3520 on 2001/06/15 by jhoule@MA_JHOULE

Other defaut buffer color.

Change 3508 on 2001/06/15 by jhoule@MA_JHOULE

Initial check-in.

Nearest filter works.

Can change filter at will.

Still quite some work to do.

Change 3500 on 2001/06/14 by jhoule@MA_JHOULE

Added renderBuffer which updates gBuffer as needed (removes need for reshape, and is more efficient).

Added special to activate/deactivate texturing, etc. under OpenGL.

Uses ImageLoader instead of TexLoader (better name).

Change 3499 on 2001/06/14 by jhoule@MA_JHOULE

Added texImage to call glTexImage2D with correct parameters.

Change 3498 on 2001/06/14 by jhoule@MA_JHOULE

mFrameBuffer is now mColorBuffer.

Default clearColor (black) added.

Change 3497 on 2001/06/14 by jhoule@MA_JHOULE

Initial check-in.

Works pretty well for PPM (other not implemented yet).

Change 3494 on 2001/06/14 by jhoule@MA_JHOULE

Removed unnecessary output.

Uses vertex arrays for rendering.

New scene.

Added reshape (which otherwise screwed driver big time).

Partly integrated texture loading.

Change 3488 on 2001/06/14 by jhoule@MA_JHOULE

changed color type from int to unsigned char

Change 3483 on 2001/06/14 by lseiler@SEILER2

Added support for mad, lrp, and floor/ceil/trunc/round (the latter only for software mode fixed point and rf, as yet)

Change 3481 on 2001/06/13 by jhoule@MA_JHOULE

Added makeScene, renderScene, renderSceneGL.

Can now compare between GL and R400Raster on screen (simple scheme, no arbitrary camera yet).

Change 3480 on 2001/06/13 by jhoule@MA_JHOULE

Added width() and height() retrievers.

Change 3476 on 2001/06/13 by smorein@smorein_r400

Texture pipe update

Change 3471 on 2001/06/13 by jhoule@MA_JHOULE

Removed ZBuffer completely: now uses Buffer from frame and depth.

Output of framebuffer in a GLUT window (warning: shrinking window screws display).

Change 3470 on 2001/06/13 by jhoule@MA_JHOULE

Finished clearColorBuffer (with default alpha value)

Change 3458 on 2001/06/12 by jhoule@MA_JHOULE

Safety check-in.

Change 3453 on 2001/06/12 by jhoule@MA_JHOULE

Added WITH_PARSER #define for looser coupling.

Change 3452 on 2001/06/12 by jhoule@MA_JHOULE

Precision turned back to 8 bits instead of the full 23.

Change 3451 on 2001/06/12 by llefebvr@llefebvre_laptop_r400

changed the delta precision of the interpolators to 8

Change 3450 on 2001/06/12 by lseiler@SEILER2

Added left and right shift, eliminated gnu floating point comparison errors due to compiler bugs.

Change 3434 on 2001/06/12 by lseiler@SEILER2

Numeric library: Quick fix for Laurent

Change 3428 on 2001/06/11 by smorein@smorein_r400

Adding a bunch of files

Change 3414 on 2001/06/11 by lseiler@SEILER2

Numeric Library with shift instructions and OFFSET macro

Change 3402 on 2001/06/11 by lseiler@SEILER2

Numeric Library: Changed operator return values from scaled to const scaled &, for example, to convert copy constructor to a const reference. Gnu compiler was somehow producing an invalid address in the copy constructor.

Change 3395 on 2001/06/11 by bbloemer@ma-jasonh

Samsung sim results.

Change 3392 on 2001/06/11 by llefebvr@llefebvre_laptop_r400

new interpolation scheme integrated with the parser files

Change 3371 on 2001/06/08 by llefebvr@llefebvre_laptop_r400

spec backup

Change 3370 on 2001/06/08 by pmitchel@pmitchel_r400_docs

added for Alex G.

Change 3360 on 2001/06/08 by lseiler@ma_lseiler

Numeric Library: Supports big fixed point numbers

Change 3330 on 2001/06/07 by llefebvr@llefebvre_laptop_r400

safety backup

Change 3313 on 2001/06/06 by lseiler@ma_lseiler

Superceded by R400_MemCtl.vsd

Change 3311 on 2001/06/06 by lseiler@ma_lseiler

Memory Controller: version 0.4, added internal interfaces and improved the Ordering Engine

Change 3285 on 2001/06/05 by jhoule@MA_JHOULE

"Solved" mipmap generation (some weird bugs, I tell ya!).
Better take a look with the GL guy... (when I know who did it)

Change 3271 on 2001/06/05 by lseiler@ma_lseiler

Numeric Library: minor fix to avoid gnu problem

Change 3259 on 2001/06/05 by lseiler@ma_lseiler

Numeric Library: This version eliminates floating point template parameters, since gnu's g++ can't handle them. As a result, fixed point OFFSET is specified differently. This version also eliminates a variety of gnu false warning messages.

Change 3250 on 2001/06/04 by bbloemer@ma-jasonh

Pinout and schedule update.

Change 3244 on 2001/06/04 by jhoule@MA_JHOULE

Added Utils (Debug.h, Globals.h, STLTricks.h)

Change 3242 on 2001/06/04 by jhoule@MA_JHOULE

Cleaner interface for faster rendering.

Added prevDT and changed default order.

Change 3234 on 2001/06/04 by jhoule@MA_JHOULE

Somehow, previous modifications didn't see the light.

New files are added, and old ones were removed.

Change 3233 on 2001/06/04 by jhoule@MA_JHOULE

Some files were included, but didn't exist anymore.
Other files were missing (showed up in dependencies).

Change 3232 on 2001/06/04 by jhoule@MA_JHOULE

Trying to solve an MSDEV-crashing issue (seems to work)

Change 3225 on 2001/06/04 by jhoule@MA_JHOULE

MouseEvent now has correct default initializer
Added displayTexture
printMipmapInfos utility function + min/max LODs limiting
renderExtendedTri now uses GL_LINE and offset (for neighboring faces)
Removed flipping of TexY (since we use GENERATE_MIPMAP_SGIS)
Cleaned some places (removed dead code and such).

Change 3222 on 2001/06/04 by jhoule@MA_JHOULE

Not using PBuffer as texture anymore (removed the ATIX call)

Change 3193 on 2001/05/30 by lseiler@ma_lseiler

Numeric package: fixed numerous gnu warnings and errors, added MIN and MAX parameters to the internal classes as an optimization

Change 3141 on 2001/05/29 by smorein@smorein_r400

added memory hub spec. I am not happy with it, and this block wins the prize for most likely to be conpleatly redesigned.

Change 3138 on 2001/05/29 by askende@andi_docs

more updates to the spec

Change 3137 on 2001/05/29 by lseiler@ma_lseiler

Numeric Library: new template parameter definitions

Change 3105 on 2001/05/25 by llefebvr@llefebvre_laptop_r400

backup sequencer

Change 3104 on 2001/05/25 by llefebvr@llefebvre_laptop_r400

new spin on RE, SC, Bary

Change 3101 on 2001/05/25 by lseiler@ma_lseiler

Tiling: latest version checks for empty primitives

Change 3098 on 2001/05/24 by jhoule@MA_JHOULE

Major reorganisation of functions.

Quite a bit a cleanup also.

Still got some problems with edges (sometimes), and automipmap doesn't seem to allocate all the levels.

Change 3091 on 2001/05/24 by lseiler@ma_lseiler

Updated RB and MC block diagrams

Change 3088 on 2001/05/24 by lseiler@ma_lseiler

Numeric Library

Change 3075 on 2001/05/23 by jhoule@MA_JHOULE

Added setGL function to set light parameters

Change 3073 on 2001/05/23 by jhoule@MA_JHOULE

A more recent version of the file (contains GL_SGIS_generate_mipmap)

Change 3071 on 2001/05/23 by smorein@smorein_r400

added base and improved area estimate

Change 3066 on 2001/05/23 by jhoule@MA_JHOULE

TexCoord4d instead of TexCoord2d (this class isn't used anymore)

Change 3065 on 2001/05/23 by jhoule@MA_JHOULE

TexCoord4d instead of 2d (no need for 2d in this app)

Change 3055 on 2001/05/23 by jhoule@MA_JHOULE

Quad + tri on top + quad on top

Change 3050 on 2001/05/22 by jhoule@MA_JHOULE

Lightmap issues

Change 3033 on 2001/05/22 by lseiler@ma_lseiler

Numeric Library: fixed default constructor problem

Change 3032 on 2001/05/22 by bbloemer@ma-jasonh

Revision from Samsung

Change 3031 on 2001/05/22 by jhoule@MA_JHOULE

Working shadow buffer (with artefacts).

Some functions don't have a prototype yet (need to refactor)

Change 3026 on 2001/05/22 by jhoule@MA_JHOULE

PBuffer, TexCoord, ATI's gl headers and LMTri added

Change 3025 on 2001/05/22 by jhoule@MA_JHOULE

Added a great many things, notably a split interface, a PBuffer, bounding box mesh rendering, and such.

Change 3024 on 2001/05/22 by jhoule@MA_JHOULE

Working texture binding (but corruptions when window is minimized)

Change 3020 on 2001/05/21 by askende@andi_docs

another revision of the shader spec

Change 3011 on 2001/05/21 by lseiler@ma_lseiler

Number Library: major revision with software mode working
New class names, lots of operations defined, test routine coded.

Change 3008 on 2001/05/21 by smorein@smorein_r400

partial update to texture spec

Change 3007 on 2001/05/21 by jhoule@MA_JHOULE

Added width/height in ctor

Change 3006 on 2001/05/21 by jhoule@MA_JHOULE

Reverted to binding the texture.

Change 3002 on 2001/05/18 by llefebvr@llefebvre_laptop_r400

stats on vertex grouping (lenght of vectors between state changes)

Change 3000 on 2001/05/18 by jhoule@MA_JHOULE

Initial CI

Change 2998 on 2001/05/18 by jhoule@MA_JHOULE

Initial CI (taken from other project)

Gives access to ATI-specific OGL functions/variables/defines.

Change 2994 on 2001/05/18 by jhoule@MA_JHOULE

Use TorontoParser for now on... (now keeps attributes)

Change 2991 on 2001/05/18 by pmitchel@FL_ALEXG

added for Alex G.

Change 2990 on 2001/05/18 by bbloemer@ma-jasonh

Info from Samsung on DDR II working documents.

Change 2987 on 2001/05/18 by jhoule@MA_JHOULE

New parser.
Keeps (and clips) attributes.
Working textures.

Change 2986 on 2001/05/18 by jhoule@MA_JHOULE

Clean slate

Change 2985 on 2001/05/18 by jhoule@MA_JHOULE

These weren't in, it seems.

Change 2984 on 2001/05/18 by jhoule@MA_JHOULE

Deleting old parser (to be replaced by new)

Change 2973 on 2001/05/18 by jhoule@MA_JHOULE

Integrated recent changes from Laurent (int turned to float, Interpolate, etc.)
texLoad works (but leaks the texture)

Change 2972 on 2001/05/18 by jhoule@MA_JHOULE

Datacol is now float

Change 2971 on 2001/05/18 by jhoule@MA_JHOULE

New parser integration (no skip ApplyTransform, loads textures, outputs every frame)

Change 2961 on 2001/05/18 by jhoule@MA_JHOULE

Parser modifications.
Since a new parser keeps all the attributes, you are now advised to use this one...
(will be checked-in soon)

Change 2960 on 2001/05/18 by bbloemer@ma-jasonh

Update for 5/22/01 call.

Change 2959 on 2001/05/18 by jhoule@MA_JHOULE

Modifications activated (new parser has arrived, so forget these)

Change 2950 on 2001/05/17 by smorein@smorein_r400

updated spec, finally checked in

Change 2934 on 2001/05/17 by llefebvr@llefebvre_laptop_r400

HZ new simulations data

Change 2930 on 2001/05/17 by lseiler@ma_lseiler

Number Library: includes sized base types (int8, uint8) etc.

Change 2929 on 2001/05/17 by jhoule@MA_JHOULE

Defautl alpha is 1

Change 2914 on 2001/05/16 by beiwang@MA_BEIWANG

L2P (Level 2 Performance evaluation tool from Toronto, Daniel Wong)

Change 2913 on 2001/05/16 by llefebvr@llefebvre_laptop_r400

there was an error in the simulation these stats are wrong. They will be replaced by accurate data soon.

Change 2912 on 2001/05/16 by beiwang@MA_BEIWANG

Cabo Sequencer detailed diagrams. Contains breakdown of R6 sequencer in first chapter.

Change 2911 on 2001/05/16 by bbloemer@ma-jasonh

MC waveforms from initial chaplin debug, by Dae Hyun Jun

Change 2901 on 2001/05/15 by lseiler@ma_lseiler

Numeric Package: fixed a compilation bug for Laurent

Change 2900 on 2001/05/15 by bbloemer@ma-jasonh

Infineon presentation.

Change 2898 on 2001/05/15 by jhoule@MA_JHOULE

Added printing functions.

Change 2897 on 2001/05/15 by jhoule@MA_JHOULE

Added lmfaces.

Change 2873 on 2001/05/15 by jhoule@MA_JHOULE

Initial check-in

Change 2872 on 2001/05/15 by jhoule@MA_JHOULE

Initial check-in

Change 2869 on 2001/05/15 by lseiler@ma_lseiler

R400 numeric library: class names changed and now supports floor() etc. for fixed point to floating point

Change 2867 on 2001/05/15 by jhoule@MA_JHOULE

Added IdxTri to project.

Change 2865 on 2001/05/15 by jhoule@MA_JHOULE

Changed number of probes from 2 to 4.

Change 2861 on 2001/05/15 by jhoule@MA_JHOULE

Integrated IdxTri in the structure.
Namespace issues solved more elegantly.

Change 2860 on 2001/05/15 by jhoule@MA_JHOULE

Integration changes.
Default constructor added (for no parameter).
Changed 3 params ctor to account for new.
Added toGL method.

Change 2857 on 2001/05/15 by llefebvr@llefebvre_laptop_r400

coarse coverage mask dumps added

Change 2856 on 2001/05/15 by jhoule@MA_JHOULE

Old typo

Change 2851 on 2001/05/14 by smorein@smorein_r400

added initial texture decompression spec

Change 2830 on 2001/05/14 by jhoule@MA_JHOULE

Ready to renderLightMap correctly.

Change 2829 on 2001/05/14 by jhoule@MA_JHOULE

Added setFrontAndBack.

Added render with skipping of face.

Change 2828 on 2001/05/14 by jhoule@MA_JHOULE

Solved gShowWire{F|f}rame discrepency.

Added setFrontAndBack.

Added render with skipping of face.

Change 2825 on 2001/05/14 by llefebvr@llefebvre_laptop_r400

backup of RE spec (not finished yet)

Change 2823 on 2001/05/14 by jhoule@MA_JHOULE

Added computeBoundingBox.

Change 2822 on 2001/05/14 by jhoule@MA_JHOULE

Added computeBoundingBox.

Removed traces.

Change 2819 on 2001/05/14 by jhoule@MA_JHOULE

Solved namespace issues.

Change 2818 on 2001/05/14 by jhoule@MA_JHOULE

Changed format for GLformat.

Added some specializations.

Change 2817 on 2001/05/14 by jhoule@MA_JHOULE

A little cleaner GLformat instread of format.

Now has accessor for data (called mData).

Change 2816 on 2001/05/14 by jhoule@MA_JHOULE

Solved namespace issues.

Change 2815 on 2001/05/14 by jhoule@MA_JHOULE

Solved namespace issues.

Change 2814 on 2001/05/14 by jhoule@MA_JHOULE

Added bounding box management in gBB (getFrontAndBack + setCamera + loadModel).

Modified renderLightMap.

Change 2813 on 2001/05/14 by jhoule@MA_JHOULE

Solved namespace issues.

Added bounding box management (getFrontAndBack + setCamera).

Added renderLightMap.

Change 2812 on 2001/05/14 by jhoule@MA_JHOULE

Added BoundingBox to project.

Change 2796 on 2001/05/11 by bbloemer@ma-jasonh

Elpida low cost memory specs and Boris' message on needed R6X MC changes to support.

Change 2790 on 2001/05/11 by bbloemer@ma-jasonh

Latest Samsung roadmap.

Change 2784 on 2001/05/11 by jhoule@MA_JHOULE

Added projectOn

Change 2781 on 2001/05/11 by jhoule@MA_JHOULE

Solved namespace issues.

Added ostream to print as argument.

Change 2780 on 2001/05/11 by jhoule@MA_JHOULE

Solved namespace issues.

Changed print method to enable ostream specification.

Change 2779 on 2001/05/11 by jhoule@MA_JHOULE

Solved namespace issues

Change 2778 on 2001/05/11 by jhoule@MA_JHOULE

Solved namespace issues

Change 2777 on 2001/05/11 by jhoule@MA_JHOULE

Solved namespace issues

Change 2776 on 2001/05/11 by jhoule@MA_JHOULE

Solved namespace issues

Change 2775 on 2001/05/11 by jhoule@MA_JHOULE

Solved namespace issues

Change 2774 on 2001/05/11 by jhoule@MA_JHOULE

Solved namespace issues

Change 2773 on 2001/05/11 by jhoule@MA_JHOULE

Not using namespace Ad anymore (moved in instead)

Change 2772 on 2001/05/10 by smorein@smorein_r400

added file

Change 2761 on 2001/05/10 by jhoule@MA_JHOULE

Added DirLight and GLMap

Change 2760 on 2001/05/10 by jhoule@MA_JHOULE

Added NewCam and Ref

Change 2759 on 2001/05/10 by jhoule@MA_JHOULE

Aesthetic

Change 2758 on 2001/05/10 by jhoule@MA_JHOULE

Namespace issues

Change 2757 on 2001/05/10 by jhoule@MA_JHOULE

Namespace issues

Change 2756 on 2001/05/10 by jhoule@MA_JHOULE

Added a few things (like drawFrustum)

Towards a working shadow map.

Change 2755 on 2001/05/10 by jhoule@MA_JHOULE

Initial CI

Change 2735 on 2001/05/10 by jhoule@MA_JHOULE

New camera using Ref and Quat.

Unfinished.

Change 2731 on 2001/05/10 by jhoule@MA_JHOULE

Added lookAt

Change 2730 on 2001/05/10 by jhoule@MA_JHOULE

Working lookAt (quite robust, actually).

W2R and R2W were a little screwed up.

Change 2729 on 2001/05/10 by jhoule@MA_JHOULE

Some cleanup. Quat(Vec, Vec, Vec) should be removed.

Change 2728 on 2001/05/10 by jhoule@MA_JHOULE

Some cleanup. Quat(Vec, Vec, Vec) should be removed.

Change 2726 on 2001/05/10 by lseiler@ma_lseiler

R400 Numeric Library, initial version
This code compiles, but doesn't yet do anything very useful

Change 2725 on 2001/05/10 by lseiler@ma_lseiler

R400 Numeric Library, initial version
This code compiles, but doesn't yet do anything very useful

Change 2724 on 2001/05/10 by lseiler@ma_lseiler

Deleted memory controller files have been moved to doc/gfx/mc

Change 2723 on 2001/05/10 by lseiler@ma_lseiler

Deleted memory controller files have been moved to doc/gfx/mc

Change 2720 on 2001/05/10 by lseiler@ma_lseiler

Memory Controller rev 0.3: lots more details, including the external interface

Change 2713 on 2001/05/09 by askende@andi_docs

more updates

Change 2712 on 2001/05/09 by askende@andi_docs

more updates

Change 2711 on 2001/05/09 by askende@andi_docs

modifications to instruction format for the shaders

Change 2700 on 2001/05/09 by askende@andi_docs

Shader specifications

Change 2697 on 2001/05/09 by jhoule@MA_JHOULE

Changed oper*(Vec) comparison for > 1e-5

(not sure if it's valid, though)

Change 2695 on 2001/05/09 by jhoule@MA_JHOULE

Quat->Mat{34}d had wrong signs in many places.

Solved using RTRendering

Change 2676 on 2001/05/09 by jhoule@MA_JHOULE

Working orbit.

Change 2667 on 2001/05/08 by jhoule@MA_JHOULE

Added orbit.

Now a child of Ref.

Change 2666 on 2001/05/08 by jhoule@MA_JHOULE

Added orbit

Change 2663 on 2001/05/08 by jhoule@MA_JHOULE

Initial CI.

Rotate shouldn't be used.

Otherwise, it seems rather robust.

Change 2662 on 2001/05/08 by jhoule@MA_JHOULE

Added constuctor for quat between two vectors (both Vec3d and Vec4d).

Weird >= instead of < gave a few warnings.

Change 2661 on 2001/05/08 by jhoule@MA_JHOULE

Added constructor for quat between two vectors (both Vec3d and Vec4d).

Change 2660 on 2001/05/08 by jhoule@MA_JHOULE

cross(Vec4d, Vec4d) wasn't in Ad namespace.

Change 2658 on 2001/05/08 by jhoule@MA_JHOULE

Quaternion transform (i.e. oper*(Quat)) added.

Change 2649 on 2001/05/08 by llefebvr@llefebvre_laptop_r400

New version of the scan converter, ready for review

Change 2637 on 2001/05/07 by markf@r400_markf

Initial Revision

Change 2636 on 2001/05/07 by markf@r400_markf

file creation

Change 2635 on 2001/05/07 by jhoule@MA_JHOULE

Somewhat working dirLights

Change 2633 on 2001/05/07 by jhoule@MA_JHOULE

Directional light class (child of Camera from AdLib)

Change 2632 on 2001/05/07 by jhoule@MA_JHOULE

Small quad over a larger one

Change 2631 on 2001/05/07 by jhoule@MA_JHOULE

Partially working dirLight funcs

Change 2630 on 2001/05/07 by jhoule@MA_JHOULE

Partially working dirLight functionality

Change 2629 on 2001/05/07 by jhoule@MA_JHOULE

Const-correctness of retrival methods

Change 2628 on 2001/05/07 by jhoule@MA_JHOULE

Const-correctness of retrival methods

Change 2626 on 2001/05/07 by jhoule@MA_JHOULE

Added DirLight (partially)

Added various draw methods (for Ref/Vec/etc.)

Change 2625 on 2001/05/07 by jhoule@MA_JHOULE

Added DirLight (partially)

Added various draw methods (for Ref/Vec/etc.)

Change 2622 on 2001/05/07 by jhoule@MA_JHOULE

Added const& for structs (Vec3d and Pt3d)

Removed load prototype

Change 2621 on 2001/05/07 by jhoule@MA_JHOULE

Added const& for the structs (Vec3d and Pt3d)

Removed load prototype.

Change 2619 on 2001/05/07 by jhoule@MA_JHOULE

*ToGL now const

Change 2618 on 2001/05/07 by jhoule@MA_JHOULE

*ToGL now const

Change 2617 on 2001/05/07 by jhoule@MA_JHOULE

Added const toGL member functions

Change 2616 on 2001/05/07 by jhoule@MA_JHOULE

Added renderScene()

Change 2615 on 2001/05/07 by jhoule@MA_JHOULE

Added renderScene()

Change 2614 on 2001/05/07 by jhoule@MA_JHOULE

Cleaning

Change 2613 on 2001/05/07 by jhoule@MA_JHOULE

Cleaning

Change 2612 on 2001/05/07 by jhoule@MA_JHOULE

Source files moved to Source directory.

Added the Scene.m file also.

Change 2611 on 2001/05/07 by jhoule@MA_JHOULE

Removed old variables (changes showInfos)

Change 2610 on 2001/05/07 by jhoule@MA_JHOULE

Better output in case of error (LoadModel)

Change 2609 on 2001/05/07 by jhoule@MA_JHOULE

Cleaned up LoadModel function.

Change 2608 on 2001/05/07 by jhoule@MA_JHOULE

Removed old objects (from previous project).

Change 2607 on 2001/05/07 by jhoule@MA_JHOULE

Removed old objects (from previous project).

Change 2606 on 2001/05/07 by jhoule@MA_JHOULE

Quad on XZ plane, face +Y

Change 2605 on 2001/05/07 by jhoule@MA_JHOULE

A HW-assisted implementation of shadow buffers using depth maps.

Change 2604 on 2001/05/07 by jhoule@MA_JHOULE

My (dated) numerical library, with more evolved stuff that grew in.

Change 2597 on 2001/05/07 by bbloemer@ma-jasonh

JEDEC document

Change 2587 on 2001/05/04 by smorein@smorein_r400

very minor update

Change 2571 on 2001/05/04 by llefebvr@llefebvre_laptop_r400

stats on 4x4 tiles HZ

Change 2570 on 2001/05/04 by llefebvr@llefebvre_laptop_r400

Spin on some specs, not all complete be checked in for safety

Change 2569 on 2001/05/04 by llefebvr@llefebvre_laptop_r400

new spin on setup

Change 2544 on 2001/05/03 by smorein@smorein_r400

Updated RBBM spec, needs more detail to be final
Added CPneeds to driver CP spec completion

Change 2524 on 2001/05/03 by lseiler@ma_lseiler

Traces: readme file for the standard set of traces and the subset of frames used within each

Change 2508 on 2001/05/02 by llefebvr@llefebvre_laptop_r400

about to change the walking algorithm and want to keep the old one...

Change 2482 on 2001/05/01 by bbloemer@ma-jasonh

300 MHz. spec

Change 2464 on 2001/05/01 by bbloemer@ma-jasonh

Samsung's 4/26/01 presentation materials.

Change 2364 on 2001/04/26 by jhoule@MA_JHOULE

Removed unnecessary traces.

Changed ParserRage128::TriangleData type to the correct Parser::dcPixelGen3dPolygons.

->

Change 2359 on 2001/04/26 by llefebvr@llefebvre_laptop_r400

updated top level spec to match RE and SC specs

Change 2358 on 2001/04/26 by lseiler@ma_lseiler

Render Backend first spec version

Change 2357 on 2001/04/26 by jhoule@MA_JHOULE

Changed super-sampling to multi-sampling.

Change 2356 on 2001/04/26 by jhoule@MA_JHOULE

A simple ReadMe file to give some infos about how to use the library.

Change 2353 on 2001/04/26 by jhoule@MA_JHOULE

A small example use of the TorontoParser.

Change 2352 on 2001/04/26 by jhoule@MA_JHOULE

Initial check-in.
It is a crude R400 rasterizer.
(I only put it up in a library... Laurent did all the actual coding)

Change 2347 on 2001/04/25 by lseiler@ma_lseiler

Added text about the RB and MC plus descriptions of some RB features

Change 2340 on 2001/04/25 by beiwang@MA_BEIWANG

DDR I Spec from JEDEC

Change 2325 on 2001/04/24 by jhoule@MA_JHOULE

Added attributes conservation when no clipping occurs.

Change 2324 on 2001/04/24 by jhoule@MA_JHOULE

Added CopySelectAttributes() inline method to VERTEX structure (used in Transform::Apply)

Change 2314 on 2001/04/23 by smorein@smorein_r400

Updated area to new area estimate, post texture path changes
checked in top level spec for larry to add to it

Change 2307 on 2001/04/23 by bbloemer@ma-jasonh

Added R200 I/O impedance control documents.

Change 2300 on 2001/04/23 by llefebvr@llefebvre_laptop_r400

New version of the rasterizer. Fixed a bug where the parameters where interpolated from the wrong vertices.

Change 2246 on 2001/04/18 by llefebvr@llefebvre_laptop_r400

new walker, rasterizer, zbuffer supporting textures and sample point movement. The movement takes center if in or any other point inside the triangle. Works well for colors but not for textures. Will experiment now with the real centroid instead.

Change 2241 on 2001/04/17 by bbloemer@ma-jasonh

Boris updated tDQSS to match JEDEC definition.

Change 2240 on 2001/04/17 by lseiler@ma_lseiler

Memory controller architectural specs and related documents

Change 2238 on 2001/04/17 by jhoule@MA_JHOULE

Added a zeroing constructor (solves clipping bug)

Change 2237 on 2001/04/17 by jhoule@MA_JHOULE

Does not skip ApplyTransform (OK except for Quake)

Change 2236 on 2001/04/17 by jhoule@MA_JHOULE

Solved issues with non-square mipmaps.

Some debugging code added (some asserts, as well as a getchar() in DEBUG when fileNotFound)

Change 2228 on 2001/04/17 by lseiler@ma_lseiler

version 0.2

Change 2179 on 2001/04/12 by beiwang@MA_BEIWANG

MC programming guide and interface documents from Boris

Change 2153 on 2001/04/11 by llefebvr@llefebvre_laptop_r400

fixed a bug in the z interpolation + new stats on R400 vs R300

Change 2137 on 2001/04/11 by bbloemer@ma-jasonh

Some info on 500 MHz. DRAM

Change 2132 on 2001/04/10 by llefebvr@llefebvre_laptop_r400

clean R400 rasterizer

Change 2113 on 2001/04/09 by jhoule@MA_JHOULE

Solved a few files misnamings.

Seems to be some legacy dependency left (rodney parser)

Change 2112 on 2001/04/09 by jhoule@MA_JHOULE

Added a few files...

Uses the new TorontoParser

Change 2111 on 2001/04/09 by jhoule@MA_JHOULE

Added size() for output

Change 2110 on 2001/04/09 by jhoule@MA_JHOULE

L2Probe hack integration (with XYAddr class)

Change 2109 on 2001/04/09 by jhoule@MA_JHOULE

Added L2Probe hack with correct flushing.

Change 2108 on 2001/04/09 by jhoule@MA_JHOULE

Moved output of gQuadPixelFlushLimit

New parser integration (Polygon vs. Triangle)

Minor memory leaks solved.

No mipmapping when specified (l == 0)

Rectangular texture w/h computation solved.

Change 2107 on 2001/04/09 by llefebvr@llefebvr_r400

revised specs and walker

Change 2097 on 2001/04/06 by bbloemer@ma-jasonh

Added April memory matrix.

Change 2082 on 2001/04/05 by jhoule@MA_JHOULE

Toronto Parser, task #6452, 2001-04-04 (retrieved by Rodney Andre and copied here on the 2001-04-05)

Change 2058 on 2001/04/04 by jhoule@MA_JHOULE

Values weren't initialized (gave wrong clipping for notcl dumps)

Change 2057 on 2001/04/04 by jhoule@MA_JHOULE

Non-activated modifs (currently testing wrong clipping)

Change 1951 on 2001/03/30 by jhoule@MA_JHOULE

Added L1_L2_INTERACTION definition for flushing issues (see flush())

Change 1950 on 2001/03/30 by jhoule@MA_JHOULE

Added L1_L2_INTERACTION preprocessor definition for flushing issues.

Changed UINT_MAX to 0xFFFFFFFF for flushed tag (but shouldn't matter after all)

Change 1948 on 2001/03/30 by jhoule@MA_JHOULE

Added new mipmap LOD calculation (which correctly changes W and H of level).

test_raster code is more versatile because of default texSize in global parameters

Change 1919 on 2001/03/29 by jhoule@MA_JHOULE

Added things to solve wrong w/h in mipID != 0

Change 1882 on 2001/03/27 by jhoule@MA_JHOULE

Sanity testing code in test_raster()

Change 1877 on 2001/03/26 by bbloemer@ma-jasonh

DDR 2 Specification

Change 1836 on 2001/03/22 by bbloemer@ma-jasonh

Moved files to under mc.

Change 1835 on 2001/03/22 by bbloemer@ma-jasonh

Moved Samsung directory from under mc.R200 Memory to under mc.

Change 1834 on 2001/03/21 by bbloemer@ma-jasonh

Hyundai 3/20 road map presentation.

Change 1826 on 2001/03/21 by jhoule@MA_JHOULE

One too many endl

Change 1825 on 2001/03/21 by jhoule@MA_JHOULE

Was a wrong FACache for L1

Change 1824 on 2001/03/21 by bbloemer@ma-jasonh

Added Micron directory and presentation.

Change 1823 on 2001/03/21 by jhoule@MA_JHOULE

MultiLevelCache added (but probaby won't be used)

Unlimit mipmap reactivated

Change 1822 on 2001/03/21 by jhoule@MA_JHOULE

Removed duplicate output

Change 1821 on 2001/03/21 by jhoule@MA_JHOULE

Moved some printing stuff.

Removed the MultiLevelCache, and hard-coded direclty (this why I changed the print from FACache to reflect L1/L2 difference)

Change 1820 on 2001/03/21 by jhoule@MA_JHOULE

Removed flushing for L2 caches (when there is an mParentCache)

Change 1818 on 2001/03/21 by jhoule@MA_JHOULE

Initial check in

Change 1817 on 2001/03/21 by jhoule@MA_JHOULE

Added "fa2" before "fa" (for parsing reasons)

Change 1815 on 2001/03/20 by jhoule@MA_JHOULE

Added setParent so that we can send pointers, and change specify the parent later.

Change 1802 on 2001/03/20 by jhoule@MA_JHOULE

Removed "tex" to insist it is a generic memory model.

Added parent cache (pCache) in constructor (default is NULL)

Change 1801 on 2001/03/20 by jhoule@MA_JHOULE

Better comments.

Removed every "tex" I found (generic memory model, not just texture).

Added parentCache in constructor.

Change 1800 on 2001/03/20 by jhoule@MA_JHOULE

Removed trace

Note: cerr << "B" << flush; is actually cerr << "B" << AbdCache::flush; Hence, it prints B1 instead of B.

Change 1799 on 2001/03/20 by jhoule@MA_JHOULE

readParent added

Change 1797 on 2001/03/20 by jhoule@MA_JHOULE

Added parentCache support (with readParent, which doesn't need to be virtual)

Change 1796 on 2001/03/20 by jhoule@MA_JHOULE

Aesthetic

Change 1779 on 2001/03/19 by jhoule@MA_JHOULE

Argument faking added.

Differences in test code.

Change 1757 on 2001/03/16 by bbloemer@ma-jasonh

Moved to Infineon directory.

Change 1756 on 2001/03/16 by bbloemer@ma-jasonh

Created Infineon directory; moved 128M spec to it. Added 2 schedules.

Change 1755 on 2001/03/16 by jhoule@MA_JHOULE

Uses an if instead of a modulo...

Will have to test with an unloaded system

Change 1752 on 2001/03/15 by jhoule@MA_JHOULE

Semicolon missing

Change 1751 on 2001/03/15 by jhoule@MA_JHOULE

Performance tweaking... (unsuccessful)

The modulo seems dreadfully slow... Is it the P4? Or simply too many calls?

Tried various inline/Int32/.size() keeping to no avail...

Change 1750 on 2001/03/15 by jhoule@MA_JHOULE

popNpush doesn't return a value (initial thought was performance, but I'm wrong)

Change 1749 on 2001/03/15 by jhoule@MA_JHOULE

Integrated CircularFIFO

Change 1748 on 2001/03/15 by jhoule@MA_JHOULE

Integrated CircularFIFO

Change 1742 on 2001/03/15 by jhoule@MA_JHOULE

Added DEFINEs so that we could easily remove statistics computation (but not output, yet)

After verification, it accounts for less than 5%.

Change 1741 on 2001/03/15 by smorein@smorein_r400

adding first real version of top level spec.

Change 1739 on 2001/03/15 by jhoule@MA_JHOULE

Removed unnecessary function template

Change 1738 on 2001/03/15 by jhoule@MA_JHOULE

Fully-associative cache

Adds a single argument (depth), which will yield direct-mapped caches when at 1.

Change 1737 on 2001/03/15 by jhoule@MA_JHOULE

Solves const-correctness

Change 1736 on 2001/03/15 by jhoule@MA_JHOULE

Solves const-correctness

Change 1735 on 2001/03/15 by jhoule@MA_JHOULE

Major cleanup in initCache.

Added readCacheConfigFile function to lighten the code.

Change 1734 on 2001/03/15 by jhoule@MA_JHOULE

Added readCacheConfigFile (solves partly the const correctness, still some work to do)

Includes FACache

Change 1733 on 2001/03/15 by jhoule@MA_JHOULE

Cleaned things up.

Constructor takes no parameter (must call setTag)

Destructor wasn't necessary (uses the compiler's)

match uses getTag (inline = incurs no cost penalty)

Added climits header

Change 1732 on 2001/03/15 by bbloemer@ma-jasonh

2M x 16b Hyundai data sheet. Couldn't find a x 32b.

Change 1727 on 2001/03/15 by jhoule@MA_JHOULE

A simple CacheLine cache that simply contains the tag (for now)

Simplifies reading and construction of other constructs.

Change 1726 on 2001/03/15 by jhoule@MA_JHOULE

Explicit virtual constructor

Change 1716 on 2001/03/14 by llefebvr@llefebvr_r400

specs for Raster block, SC, SU, HZ + stats for HZ precision

Change 1700 on 2001/03/13 by jhoule@MA_JHOULE

Added UNLIMIT_MIPMAP #if/#endif

Change 1699 on 2001/03/13 by jhoule@MA_JHOULE

Cleaner reading for default values of cache tag bits.

Reactivated the UNLIMIT thingy

Change 1698 on 2001/03/13 by jhoule@MA_JHOULE

Removed mipmap conflict workaround (changes less than 0.5%!!!)

Could be turned back on simply...

Change 1696 on 2001/03/12 by beiwang@MA_BEIWANG

Micron DesignLine: DDR SDRAM Functionality and Controller Read Data Capture

Change 1694 on 2001/03/12 by beiwang@MA_BEIWANG

Winbond info

Change 1690 on 2001/03/12 by jhoule@MA_JHOULE

Main loop reactivated (was testing caches)

Change 1689 on 2001/03/12 by jhoule@MA_JHOULE

Flushes after n quads reactivated

Change 1688 on 2001/03/12 by jhoule@MA_JHOULE

Added Mag/Min count

Change 1687 on 2001/03/12 by jhoule@MA_JHOULE

Added findIndex with baseTexAddr

Change 1686 on 2001/03/12 by jhoule@MA_JHOULE

Added findIndex with baseTexAddr as paramter choice

Change 1680 on 2001/03/09 by llefebvr@llefebvr_r400

HZ precision simulations

Change 1663 on 2001/03/08 by beiwang@MA_BEIWANG

Information related to memory controller, memory chips, roadmaps, past generation mc specs, etc

Change 1658 on 2001/03/08 by lseiler@ma_lseiler

Word files and pdf files describing the memory controller and the frame buffer formats.

Change 1657 on 2001/03/08 by lseiler@ma_lseiler

Viso files with figures for my R400 documents. These files are divided up by the type of figure, rather than by the document that uses them. The Word documents contain links to these files.

Change 1656 on 2001/03/08 by lseiler@ma_lseiler

Updated tiling test that checks for a wide range of cache sizes, swizzling the cache blocks to improve the efficiency for a 2-block cache.

Change 1655 on 2001/03/08 by jhoule@MA_JHOULE

Added another scheme (also does vertical quad spitting)

Change 1653 on 2001/03/07 by jhoule@MA_JHOULE

Came back with old output scheme

Change 1652 on 2001/03/07 by jhoule@MA_JHOULE

Added a new output scheme (untested, but straitforward)

Change 1651 on 2001/03/07 by jhoule@MA_JHOULE

Use -fq instead of -fp

Removes Steve's NO-MAG hack

Change 1641 on 2001/03/07 by pmitchel@pmitchel_r400_docs

test

Change 1640 on 2001/03/07 by pmitchel@pmitchel_r400_docs

test

Change 1639 on 2001/03/07 by pmitchel@pmitchel_r400_docs

test

Change 1618 on 2001/03/05 by pmitchel@smcs9

submit for Steve M.

Change 1605 on 2001/03/02 by jhoule@MA_JHOULE

UINT_MAX used for default flushing limit.

Wrongfully used / instead or \ for specials C characters

Change 1604 on 2001/03/02 by jhoule@MA_JHOULE

InitCache has the char* unconst (because of exception ErrFileNotFound)

Change 1603 on 2001/03/02 by jhoule@MA_JHOULE

CacheConfig integrated..

Solved a bug that never happened: use multiCache to gCacheStat->add and NOT gCache (method is overloaded)

Change 1594 on 2001/03/01 by jhoule@MA_JHOULE

Added initCache()

Change 1593 on 2001/03/01 by jhoule@MA_JHOULE

Added initCache().

Corrected a few hard-coded caches.

Change 1590 on 2001/03/01 by jhoule@MA_JHOULE

Output of flush limit (if there is one)

Change 1588 on 2001/03/01 by jhoule@MA_JHOULE

Correction caches (use a temporary multiCache2 to be removed)

Change 1585 on 2001/03/01 by jhoule@MA_JHOULE

Inverted two caches (4x4 2x4 and 4x4 4x2)

Change 1584 on 2001/03/01 by llefebvr@llefebvr_r400

revised the end graph of the spec because of an error

Change 1578 on 2001/02/28 by jhoule@MA_JHOULE

A lot of cache cleaning.

Added comments on bit settings.

Removed output in init().

Change 1577 on 2001/02/28 by jhoule@MA_JHOULE

Output sent only if CACHE_VERBOSE high enough

Change 1576 on 2001/02/28 by jhoule@MA_JHOULE

Changed the cache sizes in define (removed 2048, addd back eventually?)

Change 1572 on 2001/02/28 by jhoule@MA_JHOULE

Added offset textureFetch calls.

Integrated FLUSH_AFTER_N_QUADS scheme to arbitrarily flush the cache after a certain number of pixels.

Better comments.

Added getMipID based on Chaplin (outdated) docs.

Calls getchar() automatically at end of program.

The strncmp calls now include the \0 at the end (differentiate -f -fp)

Change 1571 on 2001/02/28 by jhoule@MA_JHOULE

Added offset parameter to textureFetch.

Added FLUSH_AFTER_N_QUADS to support arbitrary quad cache flushing.

Change 1568 on 2001/02/28 by smorein@smorein_r400

new document, pre early draft

Change 1554 on 2001/02/27 by pmitchel@pmitchel_r400_docs

cron test

Change 1553 on 2001/02/27 by pmitchel@pmitchel_r400_docs

cron test

Change 1551 on 2001/02/27 by llefebvr@llefebvr_r400

SC and RS specs

Change 1544 on 2001/02/27 by pmitchel@pmitchel_r400_docs

last test of cron

Change 1543 on 2001/02/27 by pmitchel@pmitchel_r400_docs

cron test...

Change 1540 on 2001/02/27 by pmitchel@pmitchel_r400_docs

test of cron job for syncing web site

Change 1536 on 2001/02/27 by jhoule@MA_JHOULE

Added getMipID

Change 1535 on 2001/02/27 by jhoule@MA_JHOULE

Cache sizes selection with DEFINE.

Indentation of comments.

Added 512 cache sizes to test.

Change 1534 on 2001/02/27 by jhoule@MA_JHOULE

Cache sizes DEFINE to simplify selection

Change 1533 on 2001/02/27 by jhoule@MA_JHOULE

Few corrections here and there (concerning quadBucket)

Flushing the cache after every new texture.

Change 1532 on 2001/02/27 by jhoule@MA_JHOULE

Cleaner output (total cache size is easier to see)

Change 1525 on 2001/02/26 by jhoule@MA_JHOULE

Reads returns garbage (always 0).

Change 1524 on 2001/02/26 by jhoule@MA_JHOULE

Removed traces.

quadpixel has (commented out) code for skipping computation

Change 1522 on 2001/02/26 by jhoule@MA_JHOULE

Removed and replaced by CacheSim.h (somewhat at least)

Change 1521 on 2001/02/26 by jhoule@MA_JHOULE

Added MultiCache{Stat} and QuadRearranger

Change 1520 on 2001/02/26 by jhoule@MA_JHOULE

static_casts

Commented unused arguments.

Better info on namespace explicit instantiation bug of VC++

Change 1519 on 2001/02/26 by jhoule@MA_JHOULE

Major function reorganization.

Should help reading

Change 1518 on 2001/02/26 by jhoule@MA_JHOULE

Replaces the weird CacheSimSetup.h (soon to be removed)

Change 1517 on 2001/02/26 by jhoule@MA_JHOULE

Pragma put here (should solve many occurances)

Change 1513 on 2001/02/26 by jhoule@MA_JHOULE

Integrated QuadRearranger...

A bigger triangle in test_raster

Change 1512 on 2001/02/26 by jhoule@MA_JHOULE

Condition in flushing method (wasn't checking if the quad was valid).

Some output traces.

Upper-left corner is now updated correctly (regardless of delta size)

Change 1511 on 2001/02/26 by jhoule@MA_JHOULE

Added iostream

Change 1509 on 2001/02/26 by jhoule@MA_JHOULE

Initial CI

Change 1490 on 2001/02/23 by llefebvr@llefebvr_r400

deletion of obsolete stats

Change 1489 on 2001/02/23 by llefebvr@llefebvr_r400

specs for the raster engine and scan converter

Change 1488 on 2001/02/23 by jhoule@MA_JHOULE

New, better, faster tests!

Change 1484 on 2001/02/23 by jhoule@MA_JHOULE

Removed fetch distribution for now

Change 1483 on 2001/02/23 by jhoule@MA_JHOULE

Different test, no getchar() at end

Change 1482 on 2001/02/23 by jhoule@MA_JHOULE

Size of cache in bits

Change 1476 on 2001/02/23 by jhoule@MA_JHOULE

MultiCache{Stats} used.

Added gFramesDone (useful when skipping)

Cleaner printing scheme.

Change 1475 on 2001/02/23 by jhoule@MA_JHOULE

Initial CI

Change 1474 on 2001/02/23 by jhoule@MA_JHOULE

MultiCacheStat is friend (can now read mCaches)

Change 1473 on 2001/02/23 by jhoule@MA_JHOULE

Added a few header files (and the pragma for Windows damn message!)

Print now also prints the cache info.

Removed legacy code (was commented)

Change 1468 on 2001/02/23 by jhoule@MA_JHOULE

First insertion...

Tested moderately

Change 1466 on 2001/02/23 by jhoule@MA_JHOULE

Added {unsigned} __int64 => {U}Int64

_WIN32 also used

Change 1465 on 2001/02/23 by jhoule@MA_JHOULE

baseAddr instead of texBaseAddr (could be anything, really)

Change 1464 on 2001/02/23 by jhoule@MA_JHOULE

XY addressing size reduced to 12 bits (4098 max tex size)

Now catches ErrFileNotFound exception

Change 1463 on 2001/02/23 by jhoule@MA_JHOULE

TIO command-line support

Change 1462 on 2001/02/22 by smorein@smorein_r400

Update area, release changes to raster engine

Change 1461 on 2001/02/22 by llefebvr@llefebvr_r400

SC specs and stats for the raster efficiencies...

Change 1453 on 2001/02/22 by jhoule@MA_JHOULE

Integrated -cl (cache line arrangement) and -co (cache line organization).

Added blockStr2bits function (used for both).

NOTE: this supposes X&Y addresses have 16 bits (65536 max tex size)

Change 1452 on 2001/02/22 by jhoule@MA_JHOULE

Again, some overflow issues (because of the *100)

Change 1438 on 2001/02/21 by llefebvr@llefebvr_r400

working scan converter with quad based pipes, and R300 simulation

Change 1437 on 2001/02/21 by jhoule@MA_JHOULE

Solved UInt32 overflow by typecasting to double before multiplying by bandwidth().

Should now be consistently giving the same as: avgPixelMiss*missBandwidth()

Change 1436 on 2001/02/21 by jhoule@MA_JHOULE

Commented the last getchar() (now that I have a decent shell to work on)

Change 1431 on 2001/02/21 by jhoule@MA_JHOULE

Cleaner init/term placement.

Integrated new DMCache constructor with print member function.

Change 1430 on 2001/02/21 by jhoule@MA_JHOULE

Added a print function.

Change 1429 on 2001/02/21 by jhoule@MA_JHOULE

Added new constructor (with all the bits), a str2TileDef (obsoleted by the constructor) as well as a print function.

Change 1427 on 2001/02/21 by jhoule@MA_JHOULE

Added str2TileDef (which shouldn't be used)

Added also a new constructor (with all the bits set up).

Finally added a print function.

Change 1425 on 2001/02/21 by llefebvr@llefebvr_r400

New stats on GUP efficiency

Change 1422 on 2001/02/21 by lseiler@ma_lseiler

Supports specifying separate x and y tile sizes

Change 1421 on 2001/02/21 by jhoule@MA_JHOULE

Uses left-right and bottom-top for interpolation.

Comments were erronous. Some added also.

Change 1420 on 2001/02/21 by jhoule@MA_JHOULE

x and y initialisation moved after declarations

Change 1419 on 2001/02/21 by jhoule@MA_JHOULE

texXb and texYb weren't initialized

Change 1397 on 2001/02/20 by jhoule@MA_JHOULE

Bandwidth info corrected (full precision + typo)

Change 1396 on 2001/02/20 by jhoule@MA_JHOULE

Better comments.

Legacy code removed.

Change 1391 on 2001/02/20 by jhoule@MA_JHOULE

Added total bandwidth printing

Change 1390 on 2001/02/20 by jhoule@MA_JHOULE

gFrameCount wasn't incremented in main loop.

Memory deallocation of pkt wasn't done for frame pakets.

Changed output for last frame.

Change 1387 on 2001/02/20 by jhoule@MA_JHOULE

RTTI in Release

Change 1386 on 2001/02/20 by jhoule@MA_JHOULE

RTTI in Release

Change 1385 on 2001/02/20 by jhoule@MA_JHOULE

Auto Precomp Headers in Release

Change 1381 on 2001/02/16 by jhoule@MA_JHOULE

Distributions passed through ostream instead of printf (uses temporary strings)

Change 1380 on 2001/02/16 by jhoule@MA_JHOULE

Skipping frames trace enabled

Change 1375 on 2001/02/16 by jhoule@MA_JHOULE

Accompanying documentation for the Toronto Parser.

Change 1374 on 2001/02/16 by jhoule@MA_JHOULE

Examples that came with the continuum directory.
Moved here for clarity.

Change 1373 on 2001/02/16 by jhoule@MA_JHOULE

Toronto Parser (Corrina Lee, Michael Liu et al.)
Code from Continuum (Feb. 2 2001)
Builds as a library (see ReadMe file)
Adds VTX_Texture in VERTEX (but ApplyTransform scraps infos).

Change 1372 on 2001/02/16 by jhoule@MA_JHOULE

Mark Fowler's rasterizer with QuadData structure put in by Larry Seiler.
The project builds a library (see ReadMe.txt file)

Change 1371 on 2001/02/16 by jhoule@MA_JHOULE

Initial check-in.
This does some texture cache modeling.

Change 1335 on 2001/02/13 by llefebvr@llefebvr_r400

per pixel validity testing

Change 871 on 2001/02/02 by smorein@smorein_r400

Added a bunch of new documents, also updated area

Change 830 on 2001/01/31 by llefebvr@llefebvr_r400

rasterizer with variable tile size and some pipe statistics gathering + stats on fifo deepness

Change 802 on 2001/01/30 by llefebvr@llefebvr_r400

Spreadsheet and graphs on pipe efficiency

Change 801 on 2001/01/30 by llefebvr@llefebvr_r400

working rasterizer. Not optimized only to be used to gather stats...

Change 761 on 2001/01/26 by llefebvr@llefebvr_r400

Working full precision block scan converter

Change 683 on 2001/01/23 by smorein@smorein_r400

added new document

Change 655 on 2001/01/19 by llefebvr@llefebvr_r400

updated excel render state sheet and stats

Change 654 on 2001/01/19 by llefebvr@llefebvr_r400

state changes presentation + crayola template

Change 631 on 2001/01/16 by lseiler@ma_lseiler

Updated to support direct-mapped caching and to fix bugs in counting the number of flushed cache entries

Change 625 on 2001/01/15 by llefebvr@llefebvr_r400

stats on the number of new render states per frame and total

Change 619 on 2001/01/12 by llefebvr@llefebvr_r400

completed stats on efficiency, streaks, and caching. Added an excel sheet on caching in doc/system.

Change 616 on 2001/01/11 by smorein@smorein_r400

Updates area to include some of the comments from andy gruber in his email of january 2, 2001

Change 614 on 2001/01/10 by llefebvr@llefebvr_r400

the list of possible instructions for the unified shader

Change 610 on 2001/01/08 by llefebvr@llefebvr_r400

adding data for evolva and proCDRS

Change 609 on 2001/01/08 by llefebvr@llefebvr_r400

updating statistics and adding vertex vs pixel worlkload

Change 592 on 2000/12/22 by lseiler@ma_lseiler

New files needed for the updated PM4 parser

Change 591 on 2000/12/22 by lseiler@ma_lseiler

Version used for data at 12/22/2000 meeting;
uses new parser and works with quake & ProCDRS.

Change 587 on 2000/12/19 by lseiler@ma_lseiler

Supports limits on cache size

Change 585 on 2000/12/19 by smorein@smorein_r400

new area estimate and intial re spec.

Change 575 on 2000/12/18 by lseiler@ma_lseiler

Uses class FragCache to compute cache merging

Change 571 on 2000/12/15 by llefebvr@llefebvr_r400

added cache statistics gathering class

Change 568 on 2000/12/15 by llefebvr@llefebvr_r400

render state cache statistics

Change 557 on 2000/12/13 by llefebvr@llefebvr_r400

statistics on render states along with explanatory readme file

Change 528 on 2000/12/12 by llefebvr@llefebvr_r400

revised statistics on the number of pixels drawn between state changes and 2x2 raster stamp efficiency. renderpixelsstats.doc is a document explaining how and what was gathered in the excel spreadsheets.

Change 526 on 2000/12/12 by lseiler@ma_lseiler

ReadMe files for traces that are available on \\ma_lseiler\perforce

Change 506 on 2000/12/11 by lseiler@ma_lseiler

These files describe PM4 traces and show where to find them in the sharable on my local disk. They are not checked in to perforce since they exist on the Toronto server and because they are so large.

Change 501 on 2000/12/11 by llefebvr@llefebvr_r400

Useless and obsolete. Do not consider the stats in those files as they are wrong

Change 500 on 2000/12/11 by llefebvr@llefebvr_r400

Corrupted numbers do not use...

Change 496 on 2000/12/11 by smorein@smorein_r400

added block descriptions, added tiling

Change 493 on 2000/12/08 by llefebvr@llefebvr_r400

documents explaining the stats and new stats

Change 489 on 2000/12/08 by lseiler@ma_lseiler

Program to test tile sizes in R400

Change 480 on 2000/12/07 by llefebvr@llefebvr_r400

new parser

Change 476 on 2000/12/07 by llefebvr@llefebvr_r400

submit befor switch to new parser

Change 466 on 2000/12/06 by lseiler@ma_lseiler

quadpixel() call now includes the Z slope as part of the prim parameter.

Change 465 on 2000/12/06 by lseiler@ma_lseiler

rasterization block size is now settable by variables BlockWidth and BlockHeight, defined in ew.cpp and declared external in raster.h. Formerly, the block size was fixed at 32x8.

Change 451 on 2000/12/04 by llefebvr@llefebvr_r400

creating new shadow buffer project

Change 450 on 2000/12/01 by lseiler@ma_lseiler

quadpixel() now takes object parameters

Change 448 on 2000/11/30 by llefebvr@llefebvr_r400

cleanup

Change 447 on 2000/11/30 by llefebvr@llefebvr_r400

new projcet files for perfsim using shared directory

Change 446 on 2000/11/30 by llefebvr@llefebvr_r400

creation of a shared directory

Change 445 on 2000/11/30 by llefebvr@llefebvr_r400

guard clipping added

Change 444 on 2000/11/27 by llefebvr@llefebvr_r400

stats with new 2x2 rasterizer for pixel streaks

Change 443 on 2000/11/22 by llefebvr@llefebvr_r400

stats on quads efficiency for vectors of 4,8,16,32

Change 442 on 2000/11/21 by llefebvr@llefebvr_r400

new corrected data

Change 441 on 2000/11/21 by llefebvr@llefebvr_r400

old files

Change 439 on 2000/11/21 by llefebvr@llefebvr_r400

added stats for quad efficiency and vectors of 4 quads efficiency

Change 437 on 2000/11/20 by llefebvr@llefebvr_r400

old rasteriser files

Change 436 on 2000/11/20 by llefebvr@llefebvr_r400

changed the rasterizer to a 2x2 quad based rasterizer

Change 435 on 2000/11/17 by llefebvr@llefebvr_r400

added pixel_streaks stats (pixels streaks between state changes) and raw data plus program to generate those numbers

Change 431 on 2000/11/16 by llefebvr@llefebvr_r400

Simulations of the performance of the rasterisers

Change 430 on 2000/11/15 by smorein@smorein_r400

Adding the initial versions of several specs.

Change 417 on 2000/11/09 by llefebvr@llefebvr_r400

test

Change 416 on 2000/11/09 by pmitchel@ma_shusaku

test

Change 415 on 2000/11/09 by llefebvr@llefebvr_r400

bit shifts optimisations for the precomputed tables

Change 414 on 2000/11/07 by llefebvr@llefebvr_r400

updated rng and completed documenting the whole thing

Change 413 on 2000/11/07 by llefebvr@llefebvr_r400

documentation about mipmaped noise (optimized version)

Change 412 on 2000/11/07 by llefebvr@llefebvr_r400

8 different small gradient tables instead of one identical one

Change 411 on 2000/11/06 by llefebvr@llefebvr_r400

working mipmaping using precomputed tables

Change 410 on 2000/11/06 by pmitchel@ma_shusaku

test of perforce word plugin

Change 408 on 2000/11/03 by llefebvr@llefebvr_r400

nothing done

Change 407 on 2000/11/03 by llefebvr@llefebvr_r400

added VC++ project file and removed useless files

Change 406 on 2000/11/03 by llefebvr@llefebvr_r400

not needed anymore in this folder

Change 405 on 2000/11/03 by llefebvr@llefebvr_r400

initial submission of all the files of the noise project

Change 404 on 2000/11/03 by llefebvr@llefebvr_r400

Media files needed by the noise project

Change 400 on 2000/11/01 by pmitchel@_pmitchel

Initial creation of r400 area under //depot

Change 238013 on 2005/10/06 by lseiler@lseiler_win_l_r400

Delete most unused Crayola tiling formats, which required changing addrR5xxArrayToAddr to addrCoordToAddr to distinguish z and s (always z in Xenos)

Change 210508 on 2005/02/28 by lseiler@lseiler_win_m_r400

Restore some changes made to find ADDR_64 bug

Change 210295 on 2005/02/27 by lseiler@lseiler_linux_r400

Linux gold images with 64-bit device addresses

Change 210279 on 2005/02/27 by lseiler@lseiler_win_h_r400

Verify addr_64 to uint32 conversions

Change 210266 on 2005/02/26 by lseiler@lseiler_win_h_r400

Gold images (with 32-bit device addresses) for tests that fail sim with 64-bit device addresses

Change 207027 on 2005/02/04 by lseiler@lseiler_win_m_r400

Eliminate references to addrenum

Change 205811 on 2005/01/28 by lseiler@lseiler_win_m_r400

Pad pitch and height for R5xx tilings

Change 202171 on 2005/01/05 by lseiler@lseiler_win_m_r400

update address library naming convention

Change 201801 on 2005/01/03 by lseiler@lseiler_win_m_r400

Changed address library naming convention

Change 187224 on 2004/09/10 by lseiler@lseiler_win_m_r400

Changed to 64-bit device addresses

Change 186463 on 2004/09/02 by jhoule@jhoule_r400_linux_marl

Integrated latest emulator testbench files from Xenos.

Change 180013 on 2004/07/22 by mkelly@fl_mkelly_r400_xp

copy

Change 180011 on 2004/07/22 by mkelly@fl_mkelly_r400_xp

copy from xenos, need to integ up to pele

Change 180000 on 2004/07/22 by mkelly@fl_mkelly_r400_xp

copy to r400 for integ to pele

Change 173946 on 2004/06/16 by lseiler@lseiler_linux_r400

Adding gold image for multiwrites test

Change 172559 on 2004/06/09 by lseiler@lseiler_win_m_r400

Breaking RB into RBT, RBD, and RBC

Change 172507 on 2004/06/08 by lseiler@lseiler_win_m_r400

Move RB registers to RBT, RBD, or RBC

Change 172322 on 2004/06/08 by lseiler@lseiler_win_m_r400

Split RB block some more, move RB_FETCH registers to RBC

Change 171396 on 2004/06/02 by lseiler@lseiler_win_m_r400

RB registers separated into RBT, RBD, and RBC blocks

Change 171053 on 2004/06/01 by mkelly@fl_mkelly_r400_xp

comment out until revised to work with the new MP IB packet

Change 168831 on 2004/05/19 by jhoule@jhoule_r400_linux_marl

Integrating tp_checksum and associated files to verify register write path

Change 168643 on 2004/05/18 by fliljero@fl_knarf_xenos

removed multipass test as it will not be applicable until the test itself & primlib are back integrated from Xenos.

Change 165023 on 2004/04/29 by vgoel@fl_vgoel2

order of acessing vertices was causing tessellation factor mismatches

Change 164313 on 2004/04/26 by jhoule@jhoule_r400_linux_marlboro

Reverse integrating latest TP/TC emulator code from Xenos in order to eventually get going with R600 developement.

Change 162605 on 2004/04/15 by lseiler@lseiler_win_l_r400

adding a golden image to test a failure case

Change 161203 on 2004/04/09 by lseiler@lseiler_win_l_r400

Switch r400 addr library names to r600 names

Change 158336 on 2004/03/29 by mangeshn@fl_mangeshn

edited test.

Change 158300 on 2004/03/29 by ashishs@fl_ashishs_r400_win

if Test PASSES then remove the FAIL REASON comment

Change 157701 on 2004/03/24 by lseiler@lseiler_win_l_r400

replaced old r400 names with r600 names

Change 157214 on 2004/03/22 by ashishs@fl_ashishs_r400_win

swapping the fb backgorund to match xenos

Change 157188 on 2004/03/22 by ashishs@fl_ashishs_xenos_xp

integrating the latest changes in r400

Change 156636 on 2004/03/19 by ashishs@fl_ashishs_xenos_xp2

integrating the change in r400

Change 156223 on 2004/03/18 by vgoel@fl_vgoel2

modified vertex shaders which were using uninitialzed GPRs components.

Change 155953 on 2004/03/17 by ashishs@fl_ashishs_xenos_xp2

integrating the changes from xenos back to r400

Change 155877 on 2004/03/17 by ashishs@fl_ashishs_r400_win

changing the gold path to t:\xenos\gold_r400

Change 155287 on 2004/03/15 by ashishs@fl_ashishs_r400_win

reverting the tests back to their R400 version and replacing events 25,26 and 27 with 24 as Frank had suggested that those events were removed

Change 155244 on 2004/03/15 by ashishs@fl_ashishs_r400_win2

correcting the shaders since had uninitialised exports

Change 155209 on 2004/03/15 by ashishs@fl_ashishs_xenos_xp2

files from xenos to R400

Change 155197 on 2004/03/15 by ashishs@fl_ashishs_r400_win

changed the script so that we can use the "fail reason" inside the xls

Change 155161 on 2004/03/15 by ashishs@fl_ashishs_r400_win2

same as xenos. removing some uninitialised export registers not used so as to avoid failing in hardware

Change 154507 on 2004/03/11 by ashishs@fl_ashishs_r400_win2

copying from xenos

Change 154471 on 2004/03/11 by ashishs@fl_ashishs_r400_win2

enabling the stress test (still need to add actual test data, will be done soon)

Change 154182 on 2004/03/10 by csampayo@fl_csampayo1_r400_win

Changed background color to display the same as Xenos

Change 154181 on 2004/03/10 by csampayo@fl_csampayo1_r400_win

Update from Xenos and, change background color to display the same as Xenos.

Change 154032 on 2004/03/10 by csampayo@fl_csampayo1_r400_win

Update from Xenos version

Change 153947 on 2004/03/10 by mkelly@fl_mkelly_r400_win_laptop

update

Change 153946 on 2004/03/10 by mkelly@fl_mkelly_r400_win_laptop

update

Change 153944 on 2004/03/10 by mkelly@fl_mkelly_r400_win_laptop

update

Change 153511 on 2004/03/08 by ashishs@fl_ashishs_r400_win2

adding loop export tests

Change 153356 on 2004/03/08 by ashishs@fl_ashishs_r400_win2

adding sq_debug register tests

Change 153078 on 2004/03/05 by csampayo@fl_csampayo4_r400_win

Increased image buffer size to 160 (multiple of both 40 and 32)

Change 152998 on 2004/03/05 by ashishs@fl_ashishs_r400_win2

removing TCL dump since not necessary in R400 dumps

Change 152991 on 2004/03/05 by mangeshn@fl_mangeshn

added looped pred_set_pop test checking result register behavior and clamp value

Change 152931 on 2004/03/05 by ashishs@fl_ashishs_r400_win2

adding an imnportant comment

Change 152930 on 2004/03/05 by ashishs@fl_ashishs_r400_win2

converted from xenos back to R400

Change 152915 on 2004/03/05 by mangeshn@fl_mangeshn

added test : executing pred_sete_push in a loop

Change 152907 on 2004/03/05 by ashishs@fl_ashishs_r400_win2

adding tests over from xenos

Change 152884 on 2004/03/05 by ashishs@fl_ashishs_r400_win2

adding the jump and call pred tests

Change 152880 on 2004/03/05 by ashishs@fl_ashishs_r400_win2

call pred simple test

Change 152879 on 2004/03/05 by ashishs@fl_ashishs_r400_win2

removed a comment

Change 152877 on 2004/03/05 by ashishs@fl_ashishs_r400_win2

adding another jmp pred test

Change 152848 on 2004/03/05 by mmantor@mmantor_xenos_linux_test

<fixed a bug in the loading of aluconst, back integrated removal of realtime space from aluconst and texconst mems and control logic in rbi and all hookups, altered sq_regress per carlos test set>

Change 152790 on 2004/03/04 by ashishs@fl_ashishs_r400_win2

adding another jump pred test

Change 152667 on 2004/03/04 by ashishs@fl_ashishs_r400_win2

adding IM_LOAD in all so that they don't fail in hardware

Change 152525 on 2004/03/03 by ashishs@fl_ashishs_r400_win2

correcting the shader

Change 152484 on 2004/03/03 by ashishs@fl_ashishs_r400_win

oops...corrected

Change 152483 on 2004/03/03 by ashishs@fl_ashishs_r400_win

updating comment

Change 152480 on 2004/03/03 by ashishs@fl_ashishs_r400_win

simple test to show that the conditional jump doesn't work correctly

Change 152451 on 2004/03/03 by ashishs@fl_ashishs_r400_win

just to show that the cond jmp doesn't work correctly

Change 152438 on 2004/03/03 by ashishs@fl_ashishs_r400_win

adding another cond call test

Change 152431 on 2004/03/03 by ashishs@fl_ashishs_r400_win

adding another cond call

Change 152404 on 2004/03/03 by ashishs@fl_ashishs_r400_win

finalizing the test

Change 152380 on 2004/03/03 by ashishs@fl_ashishs_r400_win

simple test to check the cond call

Change 152363 on 2004/03/03 by ashishs@fl_ashishs_r400_win

adding serialize keyword since failing in hardware

Change 151670 on 2004/02/27 by csampayo@fl_csampayo1_r400_win

Adding tests for hardware sanity checking

Change 151660 on 2004/02/27 by mkelly@fl_mkelly_r400_win_laptop

remove real time tests from regression

Change 151658 on 2004/02/27 by mkelly@fl_mkelly_r400_win_laptop

remove rts test

Change 151547 on 2004/02/27 by ashishs@fl_ashishs_r400_win2

adding new short tests

Change 151544 on 2004/02/27 by ashishs@fl_ashishs_r400_win2

adding the shortened versions to r400 also(already aded to xenos)

Change 151507 on 2004/02/27 by tmartin@tmartin_r400_win

improved test

Change 151484 on 2004/02/27 by ashishs@fl_ashishs_r400_win

correcting a small error in the script

Change 150972 on 2004/02/25 by ashishs@fl_ashishs_r400_win

Adding loop, subroutine, REP, UNTIL combo tests

Change 150971 on 2004/02/25 by ashishs@fl_ashishs_r400_win

removing xenos related settings since they were copied from xenos

Change 150803 on 2004/02/24 by ashishs@fl_ashishs_r400_win2

adding test with nested LOOP and REP and UNTIL

Change 150706 on 2004/02/24 by ashishs@fl_ashishs_r400_win2

having one outer for loop and 3 inner nested REP loops and each rep loop using the outer for loop "i"

Change 150548 on 2004/02/23 by ashishs@fl_ashishs_r400_win2

correcting the test, i was expecting the wrong answer when the test was generating the correct answer

Change 150336 on 2004/02/20 by ashishs@fl_ashishs_r400_win2

initial checkin for the test

Change 150333 on 2004/02/20 by ashishs@fl_ashishs_r400_win2

The simplest nested LOOP and REP

Change 150304 on 2004/02/20 by ashishs@fl_ashishs_r400_win2

another loop rep test

Change 150285 on 2004/02/20 by ashishs@fl_ashishs_r400_win2

correcting the test

Change 150261 on 2004/02/20 by ashishs@fl_ashishs_r400_win2

adding test to show problem

Change 150182 on 2004/02/20 by lseiler@lseiler_r400_win_marlboro1

Change address library r400 names to r600 names

Change 149968 on 2004/02/19 by mkelly@fl_mkelly_r400_win_laptop

add pipe 0 to rsp_07, test simd 1, pipes 0 - 15

Change 149850 on 2004/02/18 by csampayo@fl_csampayo1_r400_win

Adding new mem export test

Change 149824 on 2004/02/18 by mkelly@fl_mkelly_r400_win_laptop

more rsp

Change 149823 on 2004/02/18 by mkelly@fl_mkelly_r400_win_laptop

more rsp coverage

Change 149631 on 2004/02/17 by mkelly@fl_mkelly_r400_win_laptop

changed disp dim

Change 149619 on 2004/02/17 by mkelly@fl_mkelly_r400_win_laptop

Changed to 4 cases only, added predication on SIMD/PIPE combination

Change 149490 on 2004/02/17 by ashishs@fl_ashishs_r400_win2

checkin final test for LOOP REP after the instruction was fixed

Change 149487 on 2004/02/17 by smoss@smoss_xenos_linux_orl

get all .sp files

Change 149273 on 2004/02/13 by ashishs@fl_ashishs_r400_win2

test to show that the loop rep doesnt work correctly

Change 149238 on 2004/02/13 by ashishs@fl_ashishs_r400_win

removing 2 tests from depot and test_list

Change 149195 on 2004/02/13 by ashishs@fl_ashishs_r400_win

adding "serialize" in the shader to not use assembler clauses

Change 149175 on 2004/02/13 by llefebvr@llefebvr_r400_emu_montreal

Fixing this test by insertion of serialize statement.

Change 149129 on 2004/02/13 by mkelly@fl_mkelly_r400_win_laptop

fixed test bug, vertex buffer override

Change 149093 on 2004/02/13 by mkelly@fl_mkelly_r400_win_laptop

update

Change 149092 on 2004/02/13 by mkelly@fl_mkelly_r400_win_laptop

Revised test

Change 149089 on 2004/02/13 by ashishs@fl_ashishs_r400_win2

temp checkin for the cond call test, need to correct the syntax for instruction

Change 148880 on 2004/02/12 by ashishs@fl_ashishs_r400_win2

test for LOOP REP

Change 148561 on 2004/02/11 by jayw@jayw_r400_linux_marlboro3

added regrss6 targer and major reduction in db_depth_cache_raddr.v

Change 148482 on 2004/02/10 by ashishs@fl_ashishs_r400_win2

same as loop pred_03 but does the predicate checking just before the end of the loop. But also shows that the UNTIL behaves like a loop rather than REPEAT...UNTIL

Change 148470 on 2004/02/10 by ashishs@fl_ashishs_r400_win2

correcting the test

Change 148390 on 2004/02/10 by ashishs@fl_ashishs_r400_win2

just simplifying the shader to show the error

Change 148368 on 2004/02/10 by jhoule@jhoule_r400_linux_marlboro

Updates from Xenos land.
Updated align_display.h to align to 32 under R400-R600, and to 40 for Xenos.

Change 148366 on 2004/02/10 by jhoule@jhoule_r400_linux_marlboro

Integrating back from Xenos

Change 148112 on 2004/02/09 by domachi@diotargetxp

- Prevent the random selection of a mip filter if the texture format does not support mipmapping - Bugzilla 3239
- In the uber_rand test, bias the lod bias value so that it will select values closer to 0.0 more often.

Change 148062 on 2004/02/09 by ashishs@fl_ashishs_r400_win2

adding the missing retain_prev tests to the test_list

Change 148061 on 2004/02/09 by ashishs@fl_ashishs_r400_win2

currently seems problems in emulator regarding the nested loop with exit condition, so waiting till this is fixed. The test has yet to be finalised

Change 148027 on 2004/02/09 by mkelly@fl_mkelly_r400_win_laptop

update

Change 148026 on 2004/02/09 by mkelly@fl_mkelly_r400_win_laptop

update

Change 147832 on 2004/02/06 by mkelly@fl_mkelly_r400_win_laptop

finalize

Change 147734 on 2004/02/06 by csampayo@fl_csampayo2_r400

Add new export tests (for SX)

Change 147686 on 2004/02/06 by ashishs@fl_ashishs_r400_win2

test having 4 nested loops with the outer 2 loops exiting when pred condition is true

Change 147599 on 2004/02/05 by ashishs@fl_ashishs_r400_win2

exiting in nested loop , still need to add more

Change 147532 on 2004/02/05 by ashishs@fl_ashishs_r400_win2

renaming tests for better organizatiion

Change 147531 on 2004/02/05 by mkelly@fl_mkelly_r400_win_laptop

checkpoint

Change 147527 on 2004/02/05 by ashishs@fl_ashishs_r400_win2

adding loop break test when the pred condition is false

Change 147434 on 2004/02/05 by ashishs@fl_ashishs_r400_win2

updating some comments

Change 147431 on 2004/02/05 by ashishs@fl_ashishs_r400_win2

finalising the test with loop being terminated on condition turning true

Change 147424 on 2004/02/05 by ashishs@fl_ashishs_r400_win

checkin for pred loop test

Change 147423 on 2004/02/05 by mkelly@fl_mkelly_r400_win_laptop

checkpoint...

Change 147417 on 2004/02/05 by ashishs@fl_ashishs_r400_win

removed r400sc_simple_triangle_rts_01

Change 147249 on 2004/02/04 by ashishs@fl_ashishs_r400_win

adding files over from xenos for the newly created tests

Change 147151 on 2004/02/04 by mkelly@fl_mkelly_r400_win_laptop

set constant after SQ const resource setup

Change 147146 on 2004/02/04 by jayw@jayw_r400_linux_marlboro3

memory address generation fix for number of pipes. heavily commented and greatly simplified from original. 'missing' casses added too.

Change 146971 on 2004/02/03 by mkelly@fl_mkelly_r400_win_laptop

Load RT constants after changing eo_rt and ps_const

Change 146970 on 2004/02/03 by llefebvr@llefebvre_laptop_r400_emu

Removed the explicit alloc for the parameters. When doing an explicit alloc you must put all allocs as explicit which was not done in this test.

Change 146943 on 2004/02/03 by mkelly@fl_mkelly_r400_win_laptop

set constant %4

Change 146942 on 2004/02/03 by mkelly@fl_mkelly_r400_win_laptop

set constant %4

Change 146641 on 2004/01/31 by ashishs@fl_ashishs_r400_win

fixing the shaders for the FAILING tests

Change 146519 on 2004/01/30 by vromaker@vromaker_r400_linux_marlboro

- this file is written by the test so it must not be under perforce control (or it will be read-only and the test will not be able to write it)

Change 146470 on 2004/01/30 by ashishs@fl_ashishs_r400_win

re-enabling all the tests and sorting

Change 146460 on 2004/01/30 by ashishs@fl_ashishs_r400_win

correcting shaders for the these tests. Re-enabling the tests in the test_list, these tests were commented out since they didnt generate image after Laurent put the MOVA error

Change 146455 on 2004/01/30 by ashishs@fl_ashishs_r400_win

correcting shaders for the these tests. Re-enabling the tests in the test_list, these tests were commented out since they didnt generate image after Laurent put the MOVA error

Change 146454 on 2004/01/30 by ashishs@fl_ashishs_r400_win

correcting tests since the coissue and scalar shader was incorrectly interchanged causing it to execute scalar twice (needed coissue twice)

Change 146450 on 2004/01/30 by ashishs@fl_ashishs_r400_win

copying over the correct shaders from xenos

Change 146427 on 2004/01/30 by csampayo@fl_csampayo_r400

Adding more stress tests and updated test_list and Test Tracker accordingly

Change 146423 on 2004/01/30 by danh@danh_xenos_linux_orl

Added 2 additional ALU constant writes so now 4 ALU constants are written (must be a muliple of 4)

Change 146255 on 2004/01/29 by csampayo@fl_csampayo2_r400

Sort list and re-enabled the following tests since they got fixed:
r400sq_flow_control_02
r400sq_flow_control_03

Change 146253 on 2004/01/29 by csampayo@fl_csampayo2_r400

Update for mova constraints

Change 146237 on 2004/01/29 by csampayo@fl_csampayo_r400

Adjust image size to be compatible with Xenos also

Change 146020 on 2004/01/28 by csampayo@fl_csampayo_r400

Sorted list

Change 145823 on 2004/01/28 by jhoule@jhoule_r400_linux_marlboro

Squashed a warning

Change 145759 on 2004/01/27 by csampayo@fl_csampayo_r400

Remove transform since it is not really used because pixels are all being killed.

Change 145661 on 2004/01/27 by llefebvr@llefebvr_r400_emu_montreal

Fixing bad shader (was writing to r60 instead of r48 causing the use of an unintialized GPR)

Change 145541 on 2004/01/27 by smoss@smoss_xenos_linux_orl

new output path

Change 145425 on 2004/01/26 by jayw@jayw_r400_linux_marlboro3

added db_depth_mux8to1 and _muxbus8 for state consolidation and cleanup.

Change 145327 on 2004/01/26 by jhoule@jhoule_r400_linux_marlboro

Cleaner solution: tp_g1 should now use {n}s_{w}x{h} format, where n is the number of samples, and w and h are the dimension of the texture.

Samples: 4s_32x32

Change 145318 on 2004/01/26 by jhoule@jhoule_r400_linux_marlboro

Cleanup of the tp_g* tests.
tp_g1 is 1 sample
tp_g2 is 2 samples
tp_g4 is 4 samples

Change 145316 on 2004/01/26 by jhoule@jhoule_r400_linux_marlboro

Fixing test to NOT do resolve pass by default.
Also killed it from C1 (to avoid compile errors).

Change 145299 on 2004/01/26 by jayw@jayw_r400_linux_marlboro3

Added db_depth_state.v

Change 145135 on 2004/01/23 by rramsey@rramsey_xenos_linux_orl

Update yield_optimize test so it tests the logic better
Fix cfsm clause boundary detection for yield_optimize and add setting
of exsm_updating to EX_EXEC state when it is going to update

Change 145018 on 2004/01/23 by danh@danh_xenos_linux_orl

Added a SERIALIZE; after the vfetch line in the sub buf7 routine because the vfetch is writing r8 and the next instruction uses r8 as a source

Change 144978 on 2004/01/23 by georgev@devel_georgev_r400_lin2_marlboro_tott

Added tests to regression to do 2 by multisample.

Change 144967 on 2004/01/23 by georgev@devel_georgev_r400_lin2_marlboro_tott

First rev of new tests. They don't work yet.

Change 144907 on 2004/01/23 by danh@danh_xenos_linux_orl

fixed typo (changed SERIALIZE: to SERIALIZE;)

Change 144882 on 2004/01/23 by mkelly@fl_mkelly_r400_win_laptop

move placement of CP idle

Change 144877 on 2004/01/23 by mkelly@fl_mkelly_r400_win_laptop

add two more RT constants

Change 144871 on 2004/01/23 by mkelly@fl_mkelly_r400_win_laptop

update

Change 144819 on 2004/01/22 by smoss@smoss_xenos_linux_orl

added r400vgt_suppress back in after #CL144818

Change 144818 on 2004/01/22 by smoss@smoss_crayola_win

modified these from Kevin Ryan's mail to force IM_LOAD instead of type 0

Change 144802 on 2004/01/22 by csampayo@fl_csampayo_r400

Update for R400/Xenos compatibility

Change 144790 on 2004/01/22 by jayw@jayw_r400_linux_marlboro3

added msaac and msaad targets. split off db_depth_cache_flushdata.v

Change 144782 on 2004/01/22 by georgev@devel_georgev_r400_lin2_marlboro_tott

Fixed test to display resolved multitexture surface.

Change 144779 on 2004/01/22 by csampayo@fl_csampayo_r400

Adding tests and updating test list and tracker accordingly

Change 144767 on 2004/01/22 by mkelly@fl_mkelly_r400_win_laptop

update

Change 144760 on 2004/01/22 by mkelly@fl_mkelly_r400_win_laptop

update

Change 144751 on 2004/01/22 by mkelly@fl_mkelly_r400_win_laptop

update

Change 144708 on 2004/01/22 by mkelly@fl_mkelly_r400_win_laptop

Add changes learned from other RTS tests, including eo_rt, base and size writes

Change 144642 on 2004/01/22 by mangeshn@fl_mangeshn

edited test

Change 144639 on 2004/01/22 by kryan@kryan_r400_win_marlboro_XP

Modified test to setup surface_height and surface_slice registers in

RB. Also removed unnecessary code for allocation.

Change 144581 on 2004/01/22 by mangeshn@fl_mangeshn

edited test

Change 144477 on 2004/01/22 by mkelly@fl_mkelly_r400_win_laptop

wait !cp_rt_busy before each RT context write of sq interp control

Change 144382 on 2004/01/21 by mangeshn@fl_mangeshn

edited tests

Change 144173 on 2004/01/21 by mkelly@fl_mkelly_r400_win_laptop

Export the 16th parameter to the pixel shader, to guarantee gpr 15 is initialized.

Change 144055 on 2004/01/20 by jhoule@jhoule_r400_linux_marlboro

Fixed lowercase vs uppercase clamping mode

Change 143981 on 2004/01/20 by mangeshn@fl_mangeshn

edited test

Change 143915 on 2004/01/20 by csampayo@fl_csampayo_r400

Switched loading of shaders to use IM_LOAD packets

Change 143800 on 2004/01/19 by mangeshn@fl_mangeshn

changed test for MUL_PREV2 opcode to handle Src X as well as W conditions

Change 143716 on 2004/01/19 by kevino@kevino_r400_release

Updated tp_multitexture?_pix.sp shader programs to fix bugs in them.

Change 143697 on 2004/01/19 by jhoule@jhoule_r400_linux_marlboro

Fixed tc_simple_3d tests which are in the tc directory, not the tp.

Change 143645 on 2004/01/19 by tmartin@tmartin_r400_win

updated DIM field due to texture changes in the emulator

Change 143576 on 2004/01/16 by csampayo@fl_csampayo_r400

Adding more mem export tests.
Updated test_list and test tracker accordingly

Change 143469 on 2004/01/16 by tmartin@tmartin_r400_win

updated for DIM texture change

Change 143457 on 2004/01/16 by tmartin@tmartin_r400_win

updated for DIM texture change

Change 143444 on 2004/01/16 by tmartin@tmartin_r400_win

updated to work with the DIM texture change

Change 143382 on 2004/01/16 by mkelly@fl_mkelly_r400_win_laptop

SQ.SQ_CONTEXT_MISC.INST_PRED_OPTIMIZE, YEILD_OPTIMIZE enabling

Change 143378 on 2004/01/16 by mkelly@fl_mkelly_r400_win_laptop

SQ_CONTEXT_MISC.YEILD_OPTIMIZE = 0x1, otherwise, same test as r400sq_data_dep_pred_18.cpp

Change 143333 on 2004/01/15 by csampayo@fl_csampayo_r400

Adding new channel masking test

Change 143323 on 2004/01/15 by tmartin@tmartin_r400_win

moved the DIM texture field to dword 5

Change 143279 on 2004/01/15 by jhoule@jhoule_r400_linux_marlboro

Fixed test so that RSP_PIPE really driver up to 4b worth.
Added real simd as a third parameter.
Changed scene and texture to ultimately drive all 3 simds.

Change 143278 on 2004/01/15 by tmartin@tmartin_r400_win

DIM texture settings was moved to dword 5

Change 143069 on 2004/01/14 by jhoule@jhoule_r400_linux_marlboro

tp_g1 now compiles

Change 143030 on 2004/01/14 by ashishs@fl_ashishs_r400_win

disabling some tests which were causing a hang

Change 142925 on 2004/01/14 by georgev@devel_georgev_r400_lin2_marlboro_tott

No changes.

Change 142909 on 2004/01/14 by georgev@devel_georgev_r400_lin2_marlboro_tott

Fix screen sizes for ..._mask_check and ..._max_addr_2 tests

Change 142463 on 2004/01/12 by csampayo@fl_csampayo_r400

Updated shaders to be the same as _03

Change 142341 on 2004/01/12 by tmartin@tmartin_r400_win

packed vertex constants so there are a total of 96 constants referenced in the vertex shader.

Change 142026 on 2004/01/09 by jhoule@jhoule_r400_linux_marlboro

RSP test

Change 142012 on 2004/01/09 by csampayo@fl_csampayo_r400

Update to use loop index within subroutines called from inside loop

Change 141965 on 2004/01/09 by cbrennan@cbrennan_r400_release

Try again to fix the miss_stall case in a timing performant way.

Change 141950 on 2004/01/09 by rramsey@rramsey_xenos_linux_orl

Increase DB_TSTATE_SIZE to 2x num constants (64)

Change 141913 on 2004/01/09 by jhoule@jhoule_r400_linux_marlboro_reg

Adding new GetWeights testcase to verify flipping situations.

Change 141911 on 2004/01/09 by mearl@mearl_xenos_linux_orl

Updated shader so only valid texture fetches will occur.

Change 141891 on 2004/01/09 by tien@tien_r400_devel_marlboro

Moved DIM to the right place and added ANISO_BIAS

Change 141880 on 2004/01/09 by mearl@mearl_xenos_linux_orl

Updated shader so only valid texture fetches will occur.

Change 141817 on 2004/01/09 by ashishs@fl_ashishs_r400_win

disabled some tests which were causing a hang

Change 141811 on 2004/01/09 by amys@amys_xenos_linux_orl

initial add of files for sq register write/read test

Change 141789 on 2004/01/09 by mkelly@fl_mkelly_r400_win_laptop

need this

Change 141653 on 2004/01/08 by mkelly@fl_mkelly_r400_win_laptop

RSP remainder combos

Change 141609 on 2004/01/08 by kevino@kevino_r400_release

Added first set of aniso_bias testcases and fixed tp_input.v to select aniso_bias from the correct place.

Change 141593 on 2004/01/08 by tmartin@tmartin_r400_win

added a vfetch to VS so the index would be initialized for the PS

Change 141541 on 2004/01/08 by mkelly@fl_mkelly_r400_win_laptop

Simplify to one framebuffer output, instancing PrimLib only once.

Change 141538 on 2004/01/08 by ashishs@fl_ashishs_r400_win

removed some tests temporarily

Change 141537 on 2004/01/08 by cbrennan@cbrennan_r400_emu

Reverse integrate rg changes from xenos. only really has fixed directory on smoke.rg.

Change 141424 on 2004/01/08 by tmartin@tmartin_r400_win

set r2 in the VS so r0 isn't X's in the PS

Change 141321 on 2004/01/07 by rramsey@rramsey_xenos_linux_orl

increase DB_ALUCST_SIZE setting to 64

Change 141171 on 2004/01/07 by mkelly@fl_mkelly_r400_win_laptop

SIMD0, pipes 0-15 EN_RSP

Change 141149 on 2004/01/07 by mkelly@fl_mkelly_r400_win_laptop

Write 0 to DISABLE_MC included

Change 141090 on 2004/01/07 by tmartin@tmartin_r400_win

idle added after cache flush

Change 141085 on 2004/01/07 by tmartin@tmartin_r400_win

idle added after cache flush

Change 141031 on 2004/01/06 by csampayo@fl_csampayo_r400

Added new counter selects and increased FB size

Change 140975 on 2004/01/06 by jhoule@jhoule_r400_linux_marlboro

Adding tp_special test to verify some obscure situations.

First testcase added (aligner_cycle_loop) which tests the 4xEE cycling.

Change 140910 on 2004/01/06 by smoss@smoss_crayola_win

coverage changes
1) added wait for idle after register writes
2) increased number of registers for debug control
3) increased number of registers for perf

Change 140790 on 2004/01/06 by tmartin@tmartin_r400_win

added r400vc_fetch_addr_range_05

Change 140789 on 2004/01/06 by tmartin@tmartin_r400_win

changed to program a minimum of 4 constants

Change 140709 on 2004/01/05 by csampayo@fl_csampayo_r400

Update to refresh addrs reg

Change 140631 on 2004/01/05 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 140621 on 2004/01/05 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 140618 on 2004/01/05 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 140607 on 2004/01/05 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 140605 on 2004/01/05 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 140603 on 2004/01/05 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 140590 on 2004/01/05 by kevino@kevino_r400_release

Added a second instroverride aniso case as well as a 3D_minmag case in order to cover some holes that Tien found in coverage.

Change 140584 on 2004/01/05 by vromaker@vromaker_r400_linux_marlboro

- fixed shader problems

Change 140578 on 2004/01/05 by kevino@kevino_r400_release

Added a couple test cases to smoke.rg that seem to hit some types of fails others don't.
Added a multitexture stress test (e.g. 1 tex low lat, low hit, other high lat, high hit)

Change 140548 on 2004/01/05 by tmartin@tmartin_r400_win

removed an unnecessesary case so the SQ doesn't export without a fetch

Change 140536 on 2004/01/05 by mkelly@fl_mkelly_r400_win_laptop

Fix constant loading...

Change 140284 on 2003/12/30 by dclifton@dclifton_xenos_linux_orl

Fixed scalar MOVA_FLOOR opcode;

Change 139405 on 2003/12/23 by jayw@jayw_r400_linux_marlboro2

LEDA fixes, format from 3 to 2 bits, fix for zcache invalidate, some signal renames.

Change 139403 on 2003/12/23 by mkelly@fl_mkelly_r400_win_laptop

Pipe 0, SimD 0 RSP'd

Change 139275 on 2003/12/23 by smoss@smoss_xenos_linux_orl

removed failing tests

Change 139204 on 2003/12/22 by jayw@jayw_r400_linux_marlboro2

Sync up to 139017.

Change 138887 on 2003/12/19 by jhoule@jhoule_r400_linux_marlboro

Removed warnings.
Dumping of PPMs is now activated by using 'setenv tp_cubemap_dump_ppm 1'.

Change 138868 on 2003/12/19 by georgev@devel_georgev_r400_lin2_marlboro_tott

Updates and new tests.

Change 138580 on 2003/12/19 by mkelly@fl_mkelly_r400_win_laptop

update RT param writes from RT to RB

Change 138461 on 2003/12/18 by amys@amys_xenos_linux_orl

modify test such that fifo depths will be programmed to be less than the physical depths of the fifos, so test errors won't occur

Change 138459 on 2003/12/18 by mkelly@fl_mkelly_r400_win_laptop

one change.

Change 138346 on 2003/12/18 by jayw@jayw_r400_linux_marlboro2

Fix for hiz failure. wrong depth swizzled for updating quad cache.

Change 138338 on 2003/12/18 by amys@amys_xenos_linux_orl

updated to include new registers

Change 138106 on 2003/12/17 by csampayo@fl_csampayo_r400

Updating the test description

Change 138105 on 2003/12/17 by mdesai@mdesai_r400_linux

Resolved all hardware issues on bug3077.
Fixed Y & Z overflow case

Change 138098 on 2003/12/17 by csampayo@fl_csampayo_r400

Adding vtx shader non-sequential mem exports

Change 138093 on 2003/12/17 by csampayo@fl_csampayo_r400

Adding pix shader non-sequential mem exports

Change 138077 on 2003/12/17 by amys@amys_xenos_linux_orl

add/delete registers as needed to update test

Change 137953 on 2003/12/16 by tmartin@tmartin_r400_win

updated to program alu constants in groups of 4
updated to program loop constants when boolean constants are changed

Change 137842 on 2003/12/16 by csampayo@fl_csampayo_r400

Adding test with channel masking (VS,PS) and pixel kill

Change 137837 on 2003/12/16 by amys@amys_xenos_linux_orl

remove read-only and write-only registers from write/read test

Change 137799 on 2003/12/16 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 137728 on 2003/12/16 by mkelly@fl_mkelly_r400_win_laptop

update.

Change 137727 on 2003/12/16 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 137679 on 2003/12/15 by csampayo@fl_csampayo_r400

Adding new channel masking test

Change 137545 on 2003/12/15 by mkelly@fl_mkelly_r400_win_laptop

update...

Change 137540 on 2003/12/15 by mkelly@fl_mkelly_r400_win_laptop

update.

Change 137470 on 2003/12/15 by rmanapat@rmanapat_r400_sun_marlboro

Changes for TCF, TCR, TCM chicken registers

Change 137469 on 2003/12/15 by cbrennan@cbrennan_r400_emu

Turn off mipmapping and packing with interlaced textures. Also disallow interlaced stacks.

Change 137456 on 2003/12/15 by kryan@kryan_r400_win_marlboro_XP

Update test to use IM_LOAD packet to load shader programs.

This is to avoid errors found in HW when using the Type0

where the shader programs were not reloaded after the first

time they had been loaded.

Change 137453 on 2003/12/15 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 137441 on 2003/12/15 by mkelly@fl_mkelly_r400_win_laptop

Add one extra RT param

Change 137394 on 2003/12/15 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 137390 on 2003/12/15 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 137387 on 2003/12/15 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 137372 on 2003/12/14 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 137366 on 2003/12/14 by smoss@smoss_xenos_linux_orl

removing two tests that fail hw until they are resolved

Change 137203 on 2003/12/12 by kevino@kevino_r400_release

Switched over to the low lat fifos in tca, fetch fifo , and tcd.
Added tcd_ipbuf_fifo_top.v and switched overe to using 2 16x141 mems instead a 1 32x141 mems.
Put latency params in for latency fifo prog depth testcases

Change 137189 on 2003/12/12 by mangeshn@fl_mangeshn

updated test

Change 137179 on 2003/12/12 by mkelly@fl_mkelly_r400_win_laptop

change again

Change 137162 on 2003/12/12 by mkelly@fl_mkelly_r400_win_laptop

Modify eo_rt write timing

Change 137157 on 2003/12/12 by mkelly@fl_mkelly_r400_win_laptop

change eo_rt, insert constant 0 to make test match gold

Change 137138 on 2003/12/12 by mkelly@fl_mkelly_r400_win_laptop

eo_rt packet type change

Change 137111 on 2003/12/12 by mkelly@fl_mkelly_r400_win_laptop

Smiles for Dan

Change 137074 on 2003/12/11 by mkelly@fl_mkelly_r400_win_laptop

Change postion of eo_rt write and packet type

Change 137030 on 2003/12/11 by cbrennan@cbrennan_r400_emu

Flipped nibble order of DXT3A_AS_1_1_1_1 to really match dx spec this time.

Change 136941 on 2003/12/11 by csampayo@fl_csampayo_r400

Use IM_LOAD packets to load shaders

Change 136835 on 2003/12/10 by csampayo@fl_csampayo_r400

Adding channel write mask with predicate stress test

Change 136700 on 2003/12/10 by mkelly@fl_mkelly_r400_win_laptop

try this

Change 136697 on 2003/12/10 by mkelly@fl_mkelly_r400_win_laptop

update

Change 136692 on 2003/12/10 by rramsey@rramsey_xenos_linux_orl

Change eo_rt from RT packet to Ring Buffer directly

Change 136682 on 2003/12/10 by kevino@kevino_r400_release

Added low latency fifo controllers to some fifos, but have disabled them with ifdefs until the controller works for all cases. Added programmable depth for latency fifos along with corresponding test cases

Change 136643 on 2003/12/10 by amys@amys_xenos_linux_orl

move constant writes to one packet

Change 136496 on 2003/12/09 by tmartin@tmartin_r400_win

streamlined the loop constant programming

Change 136464 on 2003/12/09 by tmartin@tmartin_r400_win

set loop constants with each draw command because the boolean constant was also changing

Change 136261 on 2003/12/08 by cbrennan@cbrennan_r400_release

Convert tests to use a common function for generating a nice distribution of texture sizes.

Change 136251 on 2003/12/08 by ashishs@fl_ashishs_r400_win2

Adding tests to test_list and adding another test for AND operation

Change 136239 on 2003/12/08 by cbrennan@cbrennan_r400_emu

Add common header for tc tests which starts with a random_texsize function for better distributions of texture sizes.

Change 136208 on 2003/12/08 by tmartin@tmartin_r400_win

removed r400vc_fetch_addr_range_05

Change 136205 on 2003/12/08 by tmartin@tmartin_r400_win

added r400vc_array_size_03

Change 136202 on 2003/12/08 by tmartin@tmartin_r400_win

tests a non-zero buffer size constant that is smaller than the true buffer size. This is actually illegal programming.

Change 136181 on 2003/12/08 by tmartin@tmartin_r400_win

updated the description

Change 136168 on 2003/12/08 by ashishs@fl_ashishs_r400_win2

Adding a CL test which shows the problem in vtx kill flag with "VTX_KILL" in the CLIP_CNTL register set to "OR" mode

Change 136157 on 2003/12/08 by tmartin@tmartin_r400_win

cleaned up the test a little

Change 136109 on 2003/12/08 by jayw@jayw_r400_linux_marlboro2

depth regression

Change 136106 on 2003/12/08 by tmartin@tmartin_r400_win

fixed some mistakes

Change 136094 on 2003/12/08 by jhoule@jhoule_r400_win_lt

New test to verify cube and stack maps.

Change 136093 on 2003/12/08 by jhoule@jhoule_r400_win_lt

Updated allocate call
Some errors now correctly return error code

Change 135863 on 2003/12/05 by csampayo@fl_csampayo3

Updated test_list and test tracker status for tests:
r400sq_gpr_index_05
r400sq_gpr_index_06
r400sq_gpr_index_07
r400sq_gpr_index_08

Change 135853 on 2003/12/05 by csampayo@fl_csampayo_r400

More GPR indexing tests

Change 135793 on 2003/12/05 by tmartin@tmartin_r400_win

fixed some typos that were not affecting the final image output

Change 135651 on 2003/12/05 by ashishs@fl_ashishs_r400_win

updating the tests and the tracker for CL_BUSY. commenting out CL_BUSY from the tests

Change 135555 on 2003/12/04 by ashishs@fl_ashishs_r400_win2

Perf counters reg coverage for SX

Change 135522 on 2003/12/04 by csampayo@fl_csampayo_r400

Updated test tracker status and added to test_list the following tests:
r400sq_gpr_index_03
r400sq_gpr_index_04

Change 135512 on 2003/12/04 by csampayo@fl_csampayo_r400

Testing pixel shader GPR indexing with clamping

Change 135509 on 2003/12/04 by csampayo@fl_csampayo_r400

Updated image size to accomodate both Xenos and R500 and changed background color for better visibility

Change 135464 on 2003/12/04 by kevino@kevino_r400_linux_marlboro

Fixed some .sp files that had not gotten integrated properly.

Change 135457 on 2003/12/04 by ashishs@fl_ashishs_r400_win2

permuting the registers of SQ_DEBUG_MISC_0

Change 135436 on 2003/12/04 by cbrennan@cbrennan_r400_emu

Enabling the randomizing mip packing.

Change 135420 on 2003/12/04 by ashishs@fl_ashishs_r400_win2

Adding a test in which the alternate points clamp on address register. The SQ_DEBUG_MISC_0.DB_PROB_ON = true and SQ_DEBUG_MISC_0.DB_PROB_BREAK = false always inside the test

Need to know why the test ahngs if DB_PROB_BREAK is turned true
And also how to read the context register SQ_DEBUG_MISC_0 after wrtiing to it

Change 135378 on 2003/12/04 by ashishs@fl_ashishs_r400_win2

simple test just setting the DB_PROB_ON in this test

Change 135353 on 2003/12/04 by mkelly@mkelly_r400_win_orl

Update...

Change 135344 on 2003/12/04 by domachi@diotargetxp

Avoid creating a 1D interlaced texture. Bugzilla 3067

Change 135334 on 2003/12/04 by tien@tien_r500_emu

Turned off crippling on the "not all" cases

Change 135330 on 2003/12/04 by smburu@smburu_r400_linux_marlboro

Fixed the Randomized testcase.

Change 135306 on 2003/12/04 by mkelly@mkelly_r400_win_orl

Remove one commented test

Change 135297 on 2003/12/04 by ashishs@fl_ashishs_r400_win2

checkin in again to initalise all the other bits of that register

Change 135283 on 2003/12/04 by kevino@kevino_r400_release

Integrate of the .sp files somehow went wrong, so it looks like this one was somehow missed from checkin last time (even though it was updated in my area and not reported as opened.)

Change 135281 on 2003/12/04 by ashishs@fl_ashishs_r400_win2

checking the test back in to report problem to Kevin

Change 135270 on 2003/12/04 by kevino@kevino_r400_release

fmt61 -only version of tp_multitexture_02.cpp

Change 135188 on 2003/12/03 by csampayo@fl_csampayo3

Initial check in

Change 135140 on 2003/12/03 by kevino@kevino_r400_release

Updated tc .sp files with serialize between fetches and alu operations
Added fmt61 tests to tp_multitexture_02

Change 135048 on 2003/12/03 by ashishs@fl_ashishs_r400_win2

making the bug case true to show the problem

Change 135046 on 2003/12/03 by ashishs@fl_ashishs_r400_win2

removing the vertex buffer writing and using the index_offset feature of the VGT to access the nest packet

Change 135009 on 2003/12/03 by jhoule@jhoule_r400_linux_marlboro

Removed last failing directed test

Change 134992 on 2003/12/03 by tmartin@tmartin_r400_linux

removed some extra spaces at the end of the files that were causing compile errors on Linux

Change 134957 on 2003/12/03 by ashishs@fl_ashishs_r400_win2

Added the GPR declaration

Change 134950 on 2003/12/03 by ashishs@fl_ashishs_r400_win2

by mistake had commented out the wait_gfx_idle, so uncommenting back

Change 134949 on 2003/12/03 by ashishs@fl_ashishs_r400_win2

just switching between the 8 shaders now, earlier was just using 1 shader

Change 134947 on 2003/12/03 by kevino@kevino_r400_release

Added serialize to shader programs under tp/data/pix and vtx

Change 134946 on 2003/12/03 by ashishs@fl_ashishs_r400_win2

removing loop from all the shaders

Change 134845 on 2003/12/02 by llefebvr@llefebvr_r400_emu_montreal

moved up the seting of the VS shader

Change 134837 on 2003/12/02 by kevino@kevino_r400_release

Updated tp shader program files to be 2.0 and inserted SERIALIZED directive between fetches and ALU commands.
Updated tp_multitexture_01 and _02 tests that create their own shader programs to give them a unique name based on testname, testcase, and seed.

Change 134831 on 2003/12/02 by ashishs@fl_ashishs_r400_win2

checking in all the shaders for the test

Change 134829 on 2003/12/02 by domachi@diotargetxp

- TConst::ValidateTexDim() must return valse for a CubeMap using an Interlaced format.
- Fix tc_random and uber_rand test so that they never attempt a CubeMap-Interlaced texture.

Change 134782 on 2003/12/02 by jhoule@jhoule_r400_linux_marlboro

Added support for randomization.
Fixed size-parsing code (wasn't working).

Change 134742 on 2003/12/02 by ashishs@fl_ashishs_r400_win2

test for SQ_DEBUF_MISC_0 register, currently under review

Change 134732 on 2003/12/02 by mkelly@fl_mkelly_r400_win_laptop

move eo_rt to top

Change 134674 on 2003/12/02 by ashishs@fl_ashishs_r400_win2

changing shader....

Change 134668 on 2003/12/02 by ashishs@fl_ashishs_r400_win2

updating description for perf counters reg coverage test

Change 134607 on 2003/12/01 by ashishs@fl_ashishs_r400_win2

Finalising the test for GPR management, cannot affect image but we can see the register printed out at the end of the test, just to verify not being overwritten by anyone else

Change 134596 on 2003/12/01 by jhoule@jhoule_r400_win_lt

Updated tp_bigmaps: now works with most sizes.

Had to update the Makefile since this test uses the address library directly, and must therefore be linked against it.

Change 134580 on 2003/12/01 by jhoule@jhoule_r400_linux_marlboro

Updated text to be gray in order to distinguish between border and texture when using either black or white border color.

Change 134566 on 2003/12/01 by ashishs@fl_ashishs_r400_win2

removing the GPR declaration from the shaders

Change 134552 on 2003/12/01 by kevino@kevino_r400_linux_marlboro

Fixed tp_multitexture tests to force signed_rf_mode to never use NoZero.
Updated tp_simple_02 tests to add 4th vertex for round_point testcases. These tests can round the index up to 3, requiring the fourth vertex.

Change 134466 on 2003/12/01 by tmartin@tmartin_r400_win

removed r400vc_stress_02 from test_list for r400. will be changed and enabled for xenos

Change 134456 on 2003/12/01 by csampayo@fl_csampayo_r400

Corrected list to include tests:
r400sx_multi_chan_pos_param_pred_export_03
r400sx_multi_chan_pos_param_pred_export_04
r400sx_multi_chan_pos_param_pred_export_05
r400sx_multi_chan_pos_param_pred_export_06

Change 134449 on 2003/12/01 by mkelly@fl_mkelly_r400_win_laptop

Wait gfx idle before starting test, ie wait for constant writes to complete..

Change 134428 on 2003/12/01 by ashishs@fl_ashishs_r400_win2

Adding another render() as suggested by Laurent

Change 134406 on 2003/12/01 by amys@amys_xenos_lnxrgs_orl

corrected format for kille instruction

Change 134405 on 2003/12/01 by amys@amys_xenos_lnxrgs_orl

modified alu constant offset to be 0

Change 134238 on 2003/11/26 by csampayo@fl_csampayo_r400

Added to test_list and updated status on tracker:
r400sx_multi_chan_pos_param_pred_export_03
r400sx_multi_chan_pos_param_pred_export_04
r400sx_multi_chan_pos_param_pred_export_05
r400sx_multi_chan_pos_param_pred_export_06

Change 134235 on 2003/11/26 by csampayo@fl_csampayo2_r400

Updated for R400/Xenos image size compatibility

Change 134234 on 2003/11/26 by csampayo@fl_csampayo2_r400

New SX chan mask and pred tests

Change 134210 on 2003/11/26 by lseiler@lseiler_r400_win_marlboro

Updated slope delta

Change 134206 on 2003/11/26 by lseiler@lseiler_r400_win_marlboro

Test of Xenos quad depth accuracy

Change 134176 on 2003/11/26 by ashishs@fl_ashishs_r400_win2

Added new test to test_list

Removed bugs from the tracker. The bugs were as follows :
In the grand total in the tracker sheet E284 was 2 times, E272 and E542 were not present therby not giving a correct total and therby not matching the schedule.
Now the tracker is correct and matches the schedule total

Change 134171 on 2003/11/26 by georgev@devel_georgev_r400_lin2_marlboro_coverage_tc

Updated script for tp and tc testbenches.

Change 134167 on 2003/11/26 by ashishs@fl_ashishs_r400_win2

SQ perf register coverage test. Need to know if the HI and LOW registers can be written and read the same way as SEL registers

Change 134165 on 2003/11/26 by llefebvr@llefebvr_r400_emu_montreal

Automatic serialization was seriously broken in this test. I changed it to manual and place the serial points where they made sense.

Change 134132 on 2003/11/26 by cbrennan@cbrennan_r400_emu

updated list of directed test failures.

Change 134094 on 2003/11/26 by georgev@devel_georgev_r400_lin2_marlboro_coverage

Added to script so that tc tests don't have to be processed by hand.

Change 134092 on 2003/11/26 by ashishs@fl_ashishs_r400_win

small correction for r400

Change 134018 on 2003/11/25 by ashishs@fl_ashishs_r400_win

Test to check the SQ_GPR_MANAGEMENT register. Currently this register is being overwritten by PRIMLIB logic

Change 134012 on 2003/11/25 by lseiler@lseiler_r400_win_marlboro1

Update after changing RF16 precision

Change 133998 on 2003/11/25 by jhoule@jhoule_r400_linux_marlboro

Forgot the actual test file... [INCOMPLETE]

Change 133996 on 2003/11/25 by jhoule@jhoule_r400_linux_marlboro

Added sparse texture test to verify very large coordinates without having to generate huge maps.

The test simply allocates and renders the corners (4 x 64x64 tiles).

[INCOMPLETE!!!]

Change 133903 on 2003/11/25 by tmartin@tmartin_r400_win

added r400sq_const_map_alu_rts_01, r400sq_const_map_alu_rts_02, r400sq_const_map_fetch_rts_01.
fixed some cell formulas

Change 133899 on 2003/11/25 by tmartin@tmartin_r400_win

Uses the max number of 32 real time texture constants in a pixel shader

Change 133884 on 2003/11/25 by ashishs@fl_ashishs_r400_win

Adding the ability to sync to top and store the sync number so that it can be used to display on emails thereby keeping track. (xenos already has this ability)

Change 133847 on 2003/11/25 by rmanapat@rmanapat_r400_release

Just a update to the rg file

Change 133839 on 2003/11/25 by tmartin@tmartin_r400_win

Uses the max number of 256 real time constants in a pixel shader

Change 133817 on 2003/11/25 by tmartin@tmartin_r400_win

Uses real time streams with room for only one constant to be allocated. The pixel shader reads a constant outside of this range. This is invalid programming, but it should still be handled without stalling.

Change 133813 on 2003/11/25 by amys@amys_xenos_lnxrgs_orl

change KILLe to all lower case--otherwise assembler error when run linux

Change 133753 on 2003/11/24 by tmartin@tmartin_r400_win

added r400sq_const_map_alu_07, r400sq_const_map_fetch_01, r400sq_const_map_fetch_02, r400sq_const_map_fetch_07, r400sq_const_map_fetch_08

Change 133751 on 2003/11/24 by tmartin@tmartin_r400_win

reduced the size of the texture constant store is reduced in order to help stall the loading of the re-mapping tables.

Change 133749 on 2003/11/24 by tmartin@tmartin_r400_win

Uses 31 texture fetch constants in a PS with one vertex fetch constant. Tests all 8 contexts. The constants are reprogrammed for each context to make sure all 160 fetch constant store locations are used.

Change 133747 on 2003/11/24 by tmartin@tmartin_r400_win

Uses 31 texture fetch constants in a PS with one vertex fetch constant

Change 133746 on 2003/11/24 by tmartin@tmartin_r400_win

Stalls the loading of the alu constant re-mapping tables

Change 133738 on 2003/11/24 by ashishs@fl_ashishs_r400_win2

updating tracker and description for the latest 2 new tests added for RETAIN_PREV opcodes

Change 133735 on 2003/11/24 by ashishs@fl_ashishs_r400_win2

Adding this weeks test as well as description to the SQ test

Change 133696 on 2003/11/24 by ashishs@fl_ashishs_r400_win2

testing PREV_RETAIN instruction with all different types of data. Needed to do mem export 2 times inside the shader since the combination of 2 scalar instructions (ADD and MAX) does cover the whole of data values input to prev_retain opcode

Change 133695 on 2003/11/24 by mkelly@fl_mkelly_r400_win_laptop

Increase timeout.

Change 133672 on 2003/11/24 by tmartin@tmartin_r400_win

test all 1280 constant store locations. previously only test 1024 locations.

Change 133639 on 2003/11/24 by jhoule@jhoule_r400_linux_marlboro

Used the scalpel: major modification.

Recoded the cylinder generation in order to solve memory overflows, use less memory, allocate automatically, use triangles instead of strips (simpler than to split in multiple strips).

Killed non-relevant testcases+functions+variables.

Testcase is now parsed to determine texture size; this potentially changes the behavior of previous testcases since they weren't properly setting the height ('construct_texture_h_size' was used for both dimensions in the only useful buildLevel call). Format of the testcase is 'tex_WxH', where W and H can take any valid value for 2D textures.

Change 133613 on 2003/11/24 by domachi@diotargetxp

Ensure cubemap textures using an interlaced format use an even height

Change 133587 on 2003/11/24 by ashishs@fl_ashishs_r400_win2

precision test for retain_prev instruction

Change 133579 on 2003/11/24 by mkelly@fl_mkelly_r400_win_laptop

Fix constant loading

Change 133572 on 2003/11/24 by ashishs@fl_ashishs_r400_win2

Test for RETAIN_PREV instruction. Does have retain_prev with all different types of instructions thereby stressing the opcode.

Change 133461 on 2003/11/21 by kevino@kevino_r400_release

Modified channel mask tests to put color in register than make sure texture channel doens't overwrite it when masked out. This way, the exported color has all channels defined.

Change 133455 on 2003/11/21 by tmartin@tmartin_r400_win

updated and added r400sq_const_map_fetch_01 and 07

Change 133436 on 2003/11/21 by tmartin@tmartin_r400_win

updated descriptions

Change 133385 on 2003/11/21 by cbrennan@cbrennan_r400_emu

Cripple formats that crash emu until fixed.. Cripple tc random testcases as well for same reason.

Change 133328 on 2003/11/21 by kevino@kevino_r400_release

Made the tp_simple_02* aniso cases smaller so they won't take as log to run. The non-PC cases still hit all aniso ratios. Made the tcdenorm tests smaller as well.

Change 133290 on 2003/11/21 by jhoule@jhoule_r400_linux_marlboro

Updated list with fixes brought by border color fix.
Also, the aniso_2D got fixed somewhere as well, but can't figure out the CL.

Change 133217 on 2003/11/20 by kevino@kevino_r400_release

Made randomized test cases smaller by making prims smaller.

Change 133206 on 2003/11/20 by kevino@kevino_r400_release

Changes to make some of the tests run a bit faster.

Change 133193 on 2003/11/20 by tmartin@tmartin_r400_win

Move the VS and PS base addresses in the constant re-mapping tables to make sure the entire range can be used. Every context is used to make sure all 1024 constants are active.

Change 133192 on 2003/11/20 by tmartin@tmartin_r400_win

updated to make sure all 1024 alu constants are programed

Change 133173 on 2003/11/20 by domachi@diotargetxp

Ensure textures with interlaced formats have an even height.

Change 133167 on 2003/11/20 by cbrennan@cbrennan_r400_emu

Crippled border color until it works.

Change 133166 on 2003/11/20 by cbrennan@cbrennan_r400_emu

Shrunk random test size down.

Change 133145 on 2003/11/20 by domachi@diotargetxp

Add all avaliable texture formats.

Change 133117 on 2003/11/20 by cbrennan@cbrennan_r400_emu

Fixed test to not go above 8192 width textures.

Change 133077 on 2003/11/20 by cbrennan@cbrennan_r400_emu

Fix for stacks with border size set.

Change 133060 on 2003/11/20 by tmartin@tmartin_r400_win

disabled real time streams so context 0 would be tested

Change 133053 on 2003/11/20 by rmanapat@rmanapat_r400_release

Removed passing tests from list

Change 133048 on 2003/11/20 by ashishs@fl_ashishs_r400_win2

adding some 12 more tests in SQ from last week and this week. Also re-enabled r400sq_auto_wrapping_memories_01

Change 133047 on 2003/11/20 by amys@amys_xenos_linux_orl

Modified all cpp and shader files to pick up changes made to vc tests

Change 133041 on 2003/11/20 by ashishs@fl_ashishs_r400_win2

Not finalised the test yet. Need to know when we have all parameters having the same attributes on all the edges (same as flat shading) it just uses the param0 color of first vertex to be FLAT.

Change 133033 on 2003/11/20 by rmanapat@rmanapat_r400_release

Updated rg file...

Change 133030 on 2003/11/20 by ashishs@fl_ashishs_r400_win2

same as 2 but keeping another side as constant

Change 133024 on 2003/11/20 by ashishs@fl_ashishs_r400_win2

Testing all 16 parameters with permuting one of the parameter having consatnt attributes on one of the edge.

Change 133021 on 2003/11/20 by amys@amys_xenos_lnxrgs_orl

add SERIALIZE statment before dependent fetch or operation

Change 133017 on 2003/11/20 by amys@amys_xenos_linux_orl

replace INT_MAX with tex_map_size for pitch

Change 132956 on 2003/11/19 by kevino@kevino_r400_release

Update to rg file to put in comment about which changelist cubic_2D_MipBaseMap was fixed in

Change 132955 on 2003/11/19 by ashishs@fl_ashishs_r400_win2

changing the shaders and removing the pred optimization.

Change 132954 on 2003/11/19 by kevino@kevino_r400_release

Several fixes for emu errors (mostly timeouts) in tp_multitexture_02_stress.
Change to tcb_fetch_generator to make TF_PIPE1 a wire instead of a parameter. Apparently synthesis has some difficulty with characterizing multiple instantiations of the same module with different parameters.

Change 132942 on 2003/11/19 by ashishs@fl_ashishs_r400_win2

Final checkin for the test. The test does 16 textures while doing combinations of switching the constant attrib between parameters over 16 parameters

Change 132919 on 2003/11/19 by kevino@kevino_r400_win_marlboro

rg file of the tp_multitexture_02_stress tests failing with emu errors

Change 132906 on 2003/11/19 by kevino@kevino_r400_release

cubic_2D_MipBaseMap fix is coming soon, so commented this out in the rg file so no one else would work on it. It is a test error.

Change 132878 on 2003/11/19 by tmartin@tmartin_r400_win

more tests were added to coincide with some emulator fixes for addressing the upper end of memory.

Change 132867 on 2003/11/19 by cbrennan@cbrennan_r400_emu

Fixed precision errors on stack maps. Removed redundant code in addresser. Updated rg file with passing tests. DOWN TO 10!

Change 132842 on 2003/11/19 by chammer@chammer_xenos_linux_orl

Added changes for Xenos, enabled with `define XENOS
Includes new rb_id, edram copy mode, zplane changes.

Change 132810 on 2003/11/19 by georgev@devel_georgev_r400_lin2_marlboro_tott

Added test to cover get weights function.

Change 132793 on 2003/11/19 by ashishs@fl_ashishs_r400_win2

Testing interpolation of 1 color when 2 or more attributes of triangle are same with polymode and clipping

Change 132751 on 2003/11/18 by csampayo@fl_csampayo_r400

Fix bad shader version and logic

Change 132743 on 2003/11/18 by ashishs@fl_ashishs_r400_win2

initial checkin for the test

Change 132722 on 2003/11/18 by ashishs@fl_ashishs_r400_win2

Added 2 more tests

Change 132709 on 2003/11/18 by ashishs@fl_ashishs_r400_win2

Adding test using the same structure as before but having clip. Need to know if seperate shaders need to be input for this test or the test1 shaders would suffice

Change 132670 on 2003/11/18 by ashishs@fl_ashishs_r400_win2

test for interpolation of constant attributes

Change 132661 on 2003/11/18 by csampayo@fl_csampayo_r400

Update to predicate processing compound indices

Change 132621 on 2003/11/18 by georgev@devel_georgev_r400_lin2_marlboro_tott

Drew too many triangles that were undefined. This has been corrected.

Change 132613 on 2003/11/18 by tmartin@tmartin_r400_win

changed images sizes to 160x160 so the dimensions are divisible by 32 and 20

Change 132610 on 2003/11/18 by rmanapat@rmanapat_r400_release

Regression file that holds all failed tests from last night regression runs

Change 132596 on 2003/11/18 by cbrennan@cbrennan_r400_emu

Cripple stacks because they are known to be broken right now.

Change 132505 on 2003/11/18 by kevino@kevino_r400_release

Change to CUBE instruction to corrrespond to llefebr's chanelist 132250: "Changing emulator and tests to meet with the new cube swizzles wich now are for SRCA zzxy (instead of zzyx). Also change the assembler to accept the new swizzle code.". This is a shader program that was missed in the update.

Change 132419 on 2003/11/17 by ashishs@fl_ashishs_r400_win

Initial checkin for test with GPR management

Change 132417 on 2003/11/17 by kevino@kevino_r400_release

All tc*agp512 testcases

Change 132401 on 2003/11/17 by tmartin@tmartin_r400_win

r400sq_const_map_alu_01
r400sq_const_map_alu_03
r400sq_const_map_alu_04
r400sq_const_map_alu_05
r400sq_const_map_alu_06
r400sq_const_map_fetch_03
r400sq_const_map_fetch_04
r400sq_const_map_fetch_05
r400sq_const_map_fetch_06

Change 132400 on 2003/11/17 by tmartin@tmartin_r400_win

uses the max number of constants in a VS and PS and stresses all 8 contexts.

Change 132388 on 2003/11/17 by tmartin@tmartin_r400_win

r400sq_const_map_fetch_04 - Uses 31 tfetch constants in a PS with one vfetch constant

Change 132386 on 2003/11/17 by kevino@kevino_r400_win_marlboro

TC tests for request size = 512bit

Change 132364 on 2003/11/17 by llefebvr@llefebvr_r400_emu_montreal

The test wasn't exporting to all 4 channels causing corruption on the input VC bus.

Change 132339 on 2003/11/17 by ashishs@fl_ashishs_r400_win

Reverted back the changes to the original state. Need to talk to Carlos about the tests since error seems to be something different than what I was thinking.

Change 132313 on 2003/11/17 by ashishs@fl_ashishs_r400_win

The shaders had index swizzles which according to me causes problems on hardware. Hence this checkin fixes the same.

Change 132312 on 2003/11/17 by georgev@devel_georgev_r400_lin2_marlboro_tott

Not finished. Added to keep track of while we're looking at the TC tests.

Change 132284 on 2003/11/17 by tmartin@tmartin_r400_win

removed unused texture constants

Change 132273 on 2003/11/17 by tmartin@tmartin_r400_win

modified to program 1024 constants even though only 512 are used

Change 132270 on 2003/11/17 by ashishs@fl_ashishs_r400_win

Changing the swizzle for the tests from zzyx -> zzxy on SrcA

Change 132265 on 2003/11/17 by tmartin@tmartin_r400_win

checkpoint

Change 132250 on 2003/11/17 by llefebvr@llefebvr_r400_linux_marlboro

Changing emulator and tests to meet with the new cube swizzles wich now are for SRCA zzxy (instead of zzyx). Also change the assembler to accept the new swizzle code.

Change 132136 on 2003/11/14 by ashishs@fl_ashishs_r400_win2

test to load shaders at any chosen location. The shader doesnt wrap in this case as expected and also need to allocate sufficient memory for shader before hand. But currently aborts in emu ...need to know why.....

Change 132103 on 2003/11/14 by ashishs@fl_ashishs_r400_win2

Finally correcting the test (debugged by Carlos). Now wraps vtx and pix shader both. The vtx and pix shader have just 1 extra slot of memory more than their size and thats how they wrap everytime they try to load

Change 132054 on 2003/11/14 by kevino@kevino_r400_release

integrate from branch to tott. These changes include making several testcases use smaller textures.

Change 132040 on 2003/11/14 by georgev@devel_georgev_r400_lin2_marlboro_tott

Went to 4 (from 8) multisamples per pixel.

Change 132031 on 2003/11/14 by georgev@devel_georgev_r400_lin2_marlboro_tott

Testing center vs. centroid.

Change 132006 on 2003/11/14 by cbrennan@cbrennan_r400_emu

Randoms don't need to be that big.

Change 131991 on 2003/11/14 by mangeshn@fl_mangeshn

update to mul_prev2 opcode

Change 131984 on 2003/11/14 by mangeshn@fl_mangeshn

updated mul_prev2 opcode in the shader

Change 131980 on 2003/11/14 by mangeshn@fl_mangeshn

updated opcode for mul_prev2

Change 131977 on 2003/11/14 by tmartin@tmartin_r400_win

fixed alu constant programming to use groups of 4

Change 131975 on 2003/11/14 by llefebvr@llefebvre_laptop_r400_emu

Porting shader to 2.0.

Change 131974 on 2003/11/14 by ashishs@fl_ashishs_r400_win2

Changinng the description inside the test, to have clear description of the test intention as well as why it is FAILING

Change 131971 on 2003/11/14 by llefebvr@llefebvre_laptop_r400_emu

porting shader tests to v2.0.

Change 131970 on 2003/11/14 by ashishs@fl_ashishs_r400_win2

Test to show that currently instruction memory wrapping has some problems

Change 131965 on 2003/11/14 by tmartin@tmartin_r400_win

Move the VS and PS base addresses in the constant re-mapping tables to make sure the entire range can be used

Change 131950 on 2003/11/14 by smoss@smoss_crayola_win

add missing test

Change 131909 on 2003/11/13 by kevino@kevino_r400_release

Added tp loop testcases for 32, 64, 128BPP

Change 131907 on 2003/11/13 by mangeshn@fl_mangeshn

update - added instructions to the test

Change 131902 on 2003/11/13 by tmartin@tmartin_r400_win

Uses the max of 32 fetch constants in a VS

Change 131896 on 2003/11/13 by rmanapat@rmanapat_r400_release

This test was not doing fmt00 and fmt01 as advertised it was
actually stuck on fmt02...problem found thanks to code coverage
Now fixed

Change 131819 on 2003/11/13 by tmartin@tmartin_r400_win

fixed alu constant programming

Change 131773 on 2003/11/13 by ashishs@fl_ashishs_r400_win2

test foir pix and vtx shader wrapping together

Change 131740 on 2003/11/13 by tmartin@tmartin_r400_win

now programs alu constants in groups of 4

Change 131737 on 2003/11/13 by ashishs@fl_ashishs_r400_win2

adding test for pixel shader wrapping

Change 131719 on 2003/11/13 by kevino@kevino_r400_release

include of control bits in tp_multitexture_01

Change 131678 on 2003/11/12 by tmartin@tmartin_r400_win

corrected expected output comment

Change 131676 on 2003/11/12 by tmartin@tmartin_r400_win

vc golds

Change 131675 on 2003/11/12 by tmartin@tmartin_r400_win

vc golds

Change 131671 on 2003/11/12 by mangeshn@fl_mangeshn

preliminary check in for the first SP stress test

Change 131666 on 2003/11/12 by ashishs@fl_ashishs_r400_win2

changed the description of the test

Change 131658 on 2003/11/12 by ashishs@fl_ashishs_r400_win2

removed the simple test as its not needed anymore

Change 131657 on 2003/11/12 by ashishs@fl_ashishs_r400_win2

initial checkin for memory wrapping test for vertex shaders. We can wrap the entire memory as many number of times as we want by changing the variable NUM_SHADERS inside the test. But inorder to run the test fast on the emulator the test has been shortened and only wraps the memory once (NUM_SHADERS = 14 will start the auto wrapping)

Change 131632 on 2003/11/12 by cbrennan@cbrennan_r400_emu

Temporarily crippled border until border bugs get cleaned out.

Change 131630 on 2003/11/12 by cbrennan@cbrennan_r400_emu

Restricted memory footprint size of vol 3d to prevent blowing out memory while creating the fills.

Change 131626 on 2003/11/12 by cbrennan@cbrennan_r400_win_marlboro

texture sizes need to be at least 1.

Change 131618 on 2003/11/12 by kevino@kevino_r400_win_marlboro

Added pixel shader program for randomized case that can handle 1-4 textures, any of which can be cube mapped (info about which textures are on and which are cube mapped go into control bits). Also made vertices more contained so smaller prims are produced.

Change 131566 on 2003/11/12 by kevino@kevino_r400_win_marlboro

For randomized test case, make vertices stay within DISP_DIM ranges. Min tex size is 1 (instead of 0).

Change 131562 on 2003/11/12 by georgev@devel_georgev_r400_lin2_marlboro_tott

Added pixel kill tests

Change 131536 on 2003/11/12 by rmanapat@rmanapat_r400_release

Just touching these files...had to do a sync -f to have them come up
even though they were already there...this should fix the EMU problem
for the MaxMag1 and 2 tests

Change 131522 on 2003/11/12 by rmanapat@rmanapat_r400_release

Fixed *_rep_* testcases for this test they were EMU error now they
run to completion...

Change 131418 on 2003/11/11 by tmartin@tmartin_r400_win

r400sq_const_map_alu_03 - uses all 512 constants in a VS
r400sq_const_map_alu_04 - uses all 512 constants in a PS
r400sq_const_map_alu_05 - tests context switching when exercising the PS
r400sq_const_map_alu_06 - tests context switching when exercising the VS

Change 131395 on 2003/11/11 by ashishs@fl_ashishs_r400_win2

re-enabling the vs_memory_wrap test by Carlos since now it PASSES again in emulator

Change 131276 on 2003/11/11 by llefebvr@llefebvre_laptop_r400_emu

These shaders were all broken because the address register was not refreshed prior to use.

Change 131212 on 2003/11/11 by mangeshn@fl_mangeshn

edit - updatig test data to reflect denorms are now being flushed correctly

Change 131158 on 2003/11/10 by mangeshn@fl_mangeshn

edit - changing order of additions during calculation of the expected value

Change 131152 on 2003/11/10 by mangeshn@fl_mangeshn

edit - changed order of additions in the opcode

Change 131147 on 2003/11/10 by mangeshn@fl_mangeshn

update - to flush FP_R400_NAN

Change 131133 on 2003/11/10 by mangeshn@fl_mangeshn

update

Change 131129 on 2003/11/10 by jhoule@jhoule_r400_linux_marlboro

Added *_rep_* testcases just like CL#129941 did for tp_simple_02.

Change 131121 on 2003/11/10 by mangeshn@fl_mangeshn

update

Change 131120 on 2003/11/10 by cbrennan@cbrennan_r400_win_marlboro

Turned off AS_8, AS_8_8 and MPEG formats from tp_cubic tests.

Change 131119 on 2003/11/10 by georgev@devel_georgev_r400_lin2_marlboro_tott

Added extra texture for MaxMag tests.

Change 131114 on 2003/11/10 by cbrennan@cbrennan_r400_emu

Fix test to force tiling on with FMT_1 and FMT_1_REVERSE.

Change 131110 on 2003/11/10 by ashishs@fl_ashishs_r400_win

adding hz and rom vars

Change 131104 on 2003/11/10 by mangeshn@fl_mangeshn

update

Change 131102 on 2003/11/10 by mangeshn@fl_mangeshn

update

Change 131099 on 2003/11/10 by ashishs@fl_ashishs_r400_win2

changing the description. Also adding a var for bug tracking

Change 131084 on 2003/11/10 by ashishs@fl_ashishs_r400_win2

Adding a simple test for bug tracking

Change 131053 on 2003/11/10 by domachi@diotargetxp

Pick more suitable random range for S & T texcoord on 1/2/3D textures.

Change 131039 on 2003/11/10 by mangeshn@fl_mangeshn

update

Change 131027 on 2003/11/10 by jhoule@jhoule_r400_linux_marlboro

On second thought, this file needs more fiddling than just a simple copy...

Change 131025 on 2003/11/10 by jhoule@jhoule_r400_linux_marlboro

Adding missing file which prevents r400tc_simple_register_indirect from compiling.
Simply snatched the one from the VC directory.

Change 131006 on 2003/11/10 by domachi@diotargetxp

Add CubeMaps and StackMaps to tc_random test.

Change 130857 on 2003/11/07 by ashishs@fl_ashishs_r400_win

adding VTX_INST_BASE and PIX_INST_BASE

Change 130819 on 2003/11/07 by mangeshn@fl_mangeshn

added export to framebuffer

Change 130810 on 2003/11/07 by ashishs@fl_ashishs_r400_win

using 64 shaders

Change 130802 on 2003/11/07 by mangeshn@fl_mangeshn

update - added export to memory for Nan/Inf data tracking

Change 130793 on 2003/11/07 by cbrennan@cbrennan_r400_emu

Got rid of tp_1D_simple_01 test. No need to get rid of it from directed test report.

Change 130792 on 2003/11/07 by kevino@kevino_r400_release

Removed a couple old tests that are no longer used (and now seg fault)

Change 130791 on 2003/11/07 by cbrennan@cbrennan_r400_emu

Fix test to not ask for 1d tiled textures. illegal case.

Change 130788 on 2003/11/07 by kevino@kevino_r400_release

Added stress tests which increase the input drive starve from 0%

Change 130774 on 2003/11/07 by mangeshn@fl_mangeshn

updated tests

Change 130689 on 2003/11/07 by mangeshn@fl_mangeshn

update

Change 130686 on 2003/11/07 by kevino@kevino_r400_win_marlboro

Small change for to filter validation code for clarity (functionaly it should be identical)

Change 130673 on 2003/11/07 by kevino@kevino_r400_win_marlboro

Copy of tp_mutlitexture_02 for fmt 16

Change 130606 on 2003/11/06 by mangeshn@fl_mangeshn

update

Change 130596 on 2003/11/06 by cbrennan@cbrennan_r400_emu

Change tc_simple_1d to use denormalized coords so that it could get past a size of 8192.

Change 130591 on 2003/11/06 by cbrennan@cbrennan_r400_emu

Un crippled 1d textures.

Change 130580 on 2003/11/06 by ashishs@fl_ashishs_r400_win

reverted the test back so that it runs (need to add inst wrap which was put by carlos and check problem)

Change 130570 on 2003/11/06 by mangeshn@fl_mangeshn

update

Change 130546 on 2003/11/06 by kevino@kevino_r400_win_marlboro

1st checkin of incomplete tp_stress test that uses AGP to have long memory latency to try to fill up fifo.:w

Change 130520 on 2003/11/06 by kevino@kevino_r400_win_marlboro

Made 2D aniso test cases use smaller prims (62.5x62.5 instead of 250x250). Since these test cases are not mipmapped, and perspective is used to force the aniso ratios, I think this should leave the functionality the same.

Change 130511 on 2003/11/06 by kevino@kevino_r400_release

Added AGP version of tc_perf_2d.cpp. Also put in 600 cycle AGP latency for all of these test cases into testCaseParams.pl.

Change 130506 on 2003/11/06 by ashishs@fl_ashishs_r400_win

changing SIZE_1 var to (SIZE_1-1) since with TP_V2=3 needs the same and now since TP_V2=3 is ON by defulat (TP_Version 2 ..i think...)

Change 130420 on 2003/11/06 by mkelly@fl_mkelly_r400_win_laptop

Move constant writes to one packet..

Change 130418 on 2003/11/06 by ashishs@fl_ashishs_r400_win2

setting env var TP_V2=3 in the var file

Change 130408 on 2003/11/06 by amys@amys_xenos_linux_orl

code modified so NANs don't cause a discrepancy between windows and linux emulations

Change 130381 on 2003/11/05 by ashishs@fl_ashishs_r400_win2

correcting a small error in script

Change 130372 on 2003/11/05 by ashishs@fl_ashishs_r400_win2

enabling ROM and HZ blocks in regression scripts

Change 130359 on 2003/11/05 by mangeshn@fl_mangeshn

added coissue test for the MUL_PREV2 instruction

Change 130352 on 2003/11/05 by kevino@kevino_r400_release

Added tp_1D_simple_01, tp_border_02, and tp_simple_01_pos, tp_mipmap_smallprim_02 to tp4_tc_random.rg

Change 130344 on 2003/11/05 by mkelly@fl_mkelly_r400_win_laptop

pipe disable tests are obsolete

Change 130335 on 2003/11/05 by mangeshn@fl_mangeshn

added coissue tests for the SIN and COS instructions

Change 130307 on 2003/11/05 by tmartin@tmartin_r400_win

changed how vertex data was assigned because NaNs were causing a problem on Linux

Change 130299 on 2003/11/05 by eberger@eberger_r400_linux_marlboro

Fixed some more errors in r400rb_simple_z.cpp.

Change 130290 on 2003/11/05 by eberger@eberger_r400_linux_marlboro

Fixed one more minor error.

Change 130286 on 2003/11/05 by eberger@eberger_r400_linux_marlboro

Corrected an error in a call to the DEPTH_SURFACE constructor.

Change 130281 on 2003/11/05 by llefebvr@llefebvr_r400_emu_montreal

Fixing SRCC valid GPR valid channel.
Putting the SERIAL on the right line the the cubic pixel shader program.

Change 130280 on 2003/11/05 by eberger@eberger_r400_linux_marlboro

Changed the DEPTH_FORMAT in one test.

Change 130269 on 2003/11/05 by eberger@eberger_r400_linux_marlboro

Created a new test very similar to Xenos's bc_simple_z.cpp.

Change 130228 on 2003/11/05 by kevino@kevino_r400_win_marlboro

Version of tp_simple_02 where textures reside in AGP. Vertices & color buffer remain in framebuffer.

Change 130217 on 2003/11/04 by ashishs@fl_ashishs_r400_win2

Tests were failing due to a change in vcrg.cpp (address clamp disable), now have been fixed

Change 130215 on 2003/11/04 by ashishs@fl_ashishs_r400_win2

corrected the syntax errors in the shaders. Now the kille instruction does require a destination register

Change 130197 on 2003/11/04 by kevino@kevino_r400_win_marlboro

If tfc.validateFilter() returns false, set volume maps to point, and aniso to disabled.
Previously only set min/mag to point, and mip to point if it was linear (leave basemap otherwise).

Change 130175 on 2003/11/04 by cbrennan@cbrennan_r400_emu

turned on uber_rand test for tp4_tc.
Turned off 1d textures in known good version of tc_random tp tests.

Change 130153 on 2003/11/04 by ashishs@fl_ashishs_r400_win

changing the test for TP_V2 with adding each shader for its own combination of sample location for fetch. Also needed to goldenise the image.

Change 130133 on 2003/11/04 by ashishs@fl_ashishs_r400_win

Changing test for TP_V2. Also needed to add different shader each fetching the texture pixel from center/centroid location

Change 130102 on 2003/11/04 by tmartin@tmartin_r400_win

Tests a high ratio of pixel threads to vertex threads

Change 130101 on 2003/11/04 by tmartin@tmartin_r400_win

Tests a high ratio (3/4) of vertex threads to pixel threads

Change 130100 on 2003/11/04 by tmartin@tmartin_r400_win

Uses each thread ID more than once in order to stress the SQ

Change 130099 on 2003/11/04 by tmartin@tmartin_r400_win

delete

Change 130098 on 2003/11/04 by tmartin@tmartin_r400_win

Overflows the vertex shader reservation station

Change 130097 on 2003/11/04 by tmartin@tmartin_r400_win

Overflows the pixel shader reservation station

Change 130093 on 2003/11/04 by tmartin@tmartin_r400_win

delete

Change 130092 on 2003/11/04 by tmartin@tmartin_r400_win

delete

Change 130084 on 2003/11/04 by tmartin@tmartin_r400_win

attempts to overflows the vertex shader reservation station

Change 130083 on 2003/11/04 by mkelly@fl_mkelly_r400_win_laptop

Update test to match Xenos for debugging

Change 130044 on 2003/11/04 by ashishs@fl_ashishs_r400_win

edited the tests to have BaseMapFilter for TP_V2 =3 also needed to goldenise the tests

Change 129956 on 2003/11/03 by tmartin@tmartin_r400_win

Overflows the pixel shader reservation stations

Change 129952 on 2003/11/03 by tmartin@tmartin_r400_win

added r400sq_thread_manage_02, r400sq_thread_manage_03, r400sq_thread_manage_04

Change 129927 on 2003/11/03 by mangeshn@fl_mangeshn

added coissue tests for the max and min instructions

Change 129905 on 2003/11/03 by mangeshn@fl_mangeshn

added coissue tests for the mova and mova_floor instructions

Change 129902 on 2003/11/03 by ashishs@fl_ashishs_r400_win

changed for TP_V2

Change 129897 on 2003/11/03 by ashishs@fl_ashishs_r400_win

change for TP_V2

Change 129865 on 2003/11/03 by ashishs@fl_ashishs_r400_win

Need to change gold for this test too.

Change 129864 on 2003/11/03 by ashishs@fl_ashishs_r400_win

changed the test as per recom from Jocelyn. Also needed to update golds since the current images seem to be better

Change 129849 on 2003/11/03 by ashishs@fl_ashishs_r400_win

Changing the test as suggested by Jocelyn for the TP_V2 change. Also needed to change the gold image since earlier due to precision isssue was incorrect and now produces a more better image now.

Change 129835 on 2003/11/03 by kevino@kevino_r400_release

Removed some deug statements I had left in

Change 129833 on 2003/11/03 by kevino@kevino_r400_release

1st cut at fixing interlaced formats. Still get seg faults for tex sizes about 32x32 or so (64x64 fails)

Change 129791 on 2003/11/03 by mkelly@fl_mkelly_r400_win_laptop

Update rt state

Change 129790 on 2003/11/03 by ashishs@fl_ashishs_r400_win

updating golden.lst file

Change 129788 on 2003/11/03 by ashishs@fl_ashishs_r400_win

Adding gold for one of the CL test to check under Linux

Change 129785 on 2003/11/03 by mkelly@fl_mkelly_r400_win_laptop

Update test arrangement, sizing const and const locations.

Change 129783 on 2003/11/03 by ashishs@fl_ashishs_r400_win

changing the test so that it PASSES under TP_V2=3

Change 129780 on 2003/11/03 by mkelly@fl_mkelly_r400_win_laptop

Move RT const sizing before setting const

Change 129767 on 2003/11/03 by mkelly@fl_mkelly_r400_win_laptop

Fix test bug where RT const size is now defined before loading RT const

Change 129766 on 2003/11/03 by mkelly@fl_mkelly_r400_win_laptop

Fixed test bug.

Change 129691 on 2003/10/31 by tmartin@tmartin_r400_win

added r400sq_thread_manage_01

Change 129688 on 2003/10/31 by tmartin@tmartin_r400_win

Tests that the expected number of threads are drawn

Change 129679 on 2003/10/31 by kevino@kevino_r400_win_marlboro

Fixes for getgradients tests. Added 2D testcase. Make sp normalize gradients to 1 for display. Add testcases to rg file.

Change 129674 on 2003/10/31 by ashishs@fl_ashishs_r400_win

initial checkin for a test using SQ PERF counters

Change 129669 on 2003/10/31 by llefebvr@llefebvr_r400_emu_montreal

fixing overwrites in mem-export tests.

Change 129623 on 2003/10/31 by mangeshn@fl_mangeshn

updated test status documents

Change 129606 on 2003/10/31 by mangeshn@fl_mangeshn

added coissue tests for mul_const, add_const and sub_const instructions

Change 129604 on 2003/10/31 by ashishs@fl_ashishs_r400_win

updating test_list and tracker for recent vtx and pix index counter tests

Change 129585 on 2003/10/31 by kevino@kevino_r400_win_marlboro

Added 1D, 3D, and Cubic getgradients testcases. None work properly yet.

Change 129581 on 2003/10/31 by mangeshn@fl_mangeshn

added coissue tests for kille, killge, killgt, killne and killone instructions

Change 129576 on 2003/10/31 by tien@tien_r400_devel_marlboro

Backed out last sim.cfg checkin
Added .rg file for block perf stuff

Change 129558 on 2003/10/31 by ashishs@fl_ashishs_r400_win2

Changing these 2 tests for output_screen_xy which are currently failing in emu but the change in test doesnt cause any difference

Change 129554 on 2003/10/31 by ashishs@fl_ashishs_r400_win2

Adding OUTPUT_SCREEN_XY to all these tests since they has PARAM_GEN as 1. Verified after regressing these tests.

Change 129547 on 2003/10/31 by tmartin@tmartin_r400_win

added r400sq_fetch_arb_01 and 02

Change 129536 on 2003/10/31 by amys@amys_xenos_linux_orl

modify tests as per r400vc tests

Change 129498 on 2003/10/30 by tmartin@tmartin_r400_win

fetch arbitration tests

Change 129447 on 2003/10/30 by mangeshn@fl_mangeshn

added coissue tests for mul_prev, add_prev and sub_prev

Change 129444 on 2003/10/30 by llefebvr@llefebvr_r400_linux_marlboro

Fixing dangling wires in the sq related to performance module.
Fixing shader due to Kill opcode assembler change.
Fixing trakcer problem in the TB_SQSP when autocount vtx is on.

Change 129377 on 2003/10/30 by mkelly@fl_mkelly_r400_win_laptop

Debug aid

Change 129349 on 2003/10/30 by mkelly@fl_mkelly_r400_win_laptop

Comment out RT and rearrange a bit for full chip test.

Change 129319 on 2003/10/29 by ashishs@fl_ashishs_r400_win

setting the mip filter to BaseMap as suggested by Jocelyn for TP_V2=3 to get PASSING

Change 129313 on 2003/10/29 by ashishs@fl_ashishs_r400_win2

testing the pix counter when rendering a triangle. Understanding the way pix counter is incremented (pix shader hit) when rendering a pixels in a triangle.

Change 129283 on 2003/10/29 by mangeshn@fl_mangeshn

changed to scalar

Change 129272 on 2003/10/29 by mangeshn@fl_mangeshn

adedd coissue tests for add, sub and mul

Change 129261 on 2003/10/29 by mkelly@fl_mkelly_r400_win_laptop

Smaller test for full chip debugging...

Change 129245 on 2003/10/29 by kevino@kevino_r400_linux_marlboro

Fixed a couple typos that left the stack depth > 64 and were causing emu errors.
Added .rg files for tp4_tc and tc for the tp_multitexture_02 stack map tests

Change 129223 on 2003/10/29 by mkelly@fl_mkelly_r400_win_laptop

Change a logic | to an add in address calc.

Change 129216 on 2003/10/29 by kevino@kevino_r400_win_marlboro

Replaced error messages about base and mip maps not being allocated with warnings not containing the word "error".

Change 129130 on 2003/10/29 by kevino@kevino_r400_win_marlboro

Updated some more of the stackmap testcases to limit the stack depth to 64

Change 129079 on 2003/10/28 by tmartin@tmartin_r400_win

updated to work when TP_V2 is 3

Change 129064 on 2003/10/28 by kevino@kevino_r400_win_marlboro

Removed a couple hacks from tp_mt_tcfunc_stack_clamp.h that I had been using for testing.

Change 129061 on 2003/10/28 by georgev@devel_georgev_r400_lin2_marlboro_tott

First revision.

Change 129060 on 2003/10/28 by kevino@kevino_r400_win_marlboro

Fixed up some of the stack filter and clamp tests. Added some new border size test cases.

Change 129058 on 2003/10/28 by georgev@devel_georgev_r400_lin2_marlboro_tott

Check in for Laurant to look at it.

Change 129023 on 2003/10/28 by mkelly@fl_mkelly_r400_win_laptop

Add sc_rcq.dmp

Change 129002 on 2003/10/28 by mangeshn@fl_mangeshn

added coissue tests for sete, setne, setge, setgt and pred_set_restore instructions

Change 128998 on 2003/10/28 by ashishs@fl_ashishs_r400_win2

Finalising the test with vsisr_cont settings, gen vtx and pix counters and exporting them to memory. Also toggling between the destination register for the pix counter between r1-r15 verey 64 points of total 4224 points

Change 128979 on 2003/10/28 by mangeshn@fl_mangeshn

added simple test for kille register export

Change 128964 on 2003/10/28 by kevino@kevino_r400_win_marlboro

Added StackMap testcases to tp_multitexture_02 that are replicas of the 3D testcases.
Have not yet gone through the results to make sure that the testcases are behaving as expected.

Change 128954 on 2003/10/28 by kevino@kevino_r400_win_marlboro

Added tp_stack test. Current testcases only test getcomplod and getgradients.

Change 128943 on 2003/10/28 by mkelly@fl_mkelly_r400_win_laptop

Remove unnecessary constant write...

Change 128936 on 2003/10/28 by ashishs@fl_ashishs_r400_win2

Test to export pix index count to all GPR's (only possible r1-r15) Using index "i" in pix shader so that just using one shader the shader can dynamically change the destination register which it is using, thereby just needing one shader for pix shader as well as testing indexing on gpr's as well as output to all gpr's

Change 128884 on 2003/10/28 by kevino@kevino_r400_win_marlboro

uncommented remove of ppm files

Change 128881 on 2003/10/28 by smoss@smoss_xenos_linux_orl

updated for primlib files

Change 128876 on 2003/10/28 by mkelly@fl_mkelly_r400_win_laptop

update...

Change 128875 on 2003/10/28 by mkelly@fl_mkelly_r400_win_laptop

update

Change 128848 on 2003/10/27 by georgev@devel_georgev_r400_lin2_marlboro_tott

Added two debug registers.

Change 128833 on 2003/10/27 by ashishs@fl_ashishs_r400_win2

input the reg index in the test. The test doesnt output on higher index values like 4,5,6.... need to know the reason

Change 128771 on 2003/10/27 by mkelly@fl_mkelly_r400_win_laptop

First HZ test, now need to migrate it to Xenos...

Change 128769 on 2003/10/27 by mkelly@fl_mkelly_r400_win_laptop

Add handling for HZ tests

Change 128768 on 2003/10/27 by kevino@kevino_r400_win_marlboro

Updates to getcomplod and getgradients testcases. The lod and gradients come out from the emu differently if the opcode is 2D versus StackMap. Since R=0.0 for all vertices, I would not expect this difference.

Change 128765 on 2003/10/27 by ashishs@fl_ashishs_r400_win

with help from Jocelyn modified the test so that works with TP_V2=3. Please see below for desc from Jocelyn

OK, from what I could observe, you end up hitting mip level 1 for some quads.
Adding the following lines right before setting the constants in the render_state solves your issues:

```
    // Set BASE_MAP filter
    point_texture_constant0.setMIP_FILTER(TFetchConst::Mip_BaseMap);
    point_texture_constant1.setMIP_FILTER(TFetchConst::Mip_BaseMap);
    ...and so on
```

Change 128682 on 2003/10/27 by kevino@kevino_r400_win_marlboro

Shader programs for getcomplod and getgrad for texture stacks. New cpp file that uses them not checked in yet.

Change 128680 on 2003/10/27 by ashishs@fl_ashishs_r400_win

changed the test so that st could bve changed easily

Change 128677 on 2003/10/27 by domachi@diotargetxp

Fix proper range for LOD_BIAS. SetLODBias should be set as a float.

Change 128672 on 2003/10/27 by amys@amys_xenos_linux_orl

converted vc tests to tc tests

Change 128618 on 2003/10/27 by mkelly@fl_mkelly_r400_win_laptop

Fix syntax error from last update.

Change 128617 on 2003/10/27 by mkelly@fl_mkelly_r400_win_laptop

Fix syntax error from last update.

Change 128616 on 2003/10/27 by mkelly@fl_mkelly_r400_win_laptop

Fix wrong CP word type.

Change 128498 on 2003/10/24 by ashishs@fl_ashishs_r400_win2

initial checkin for the test. The test uses vsisr_cont with gen vtx and pix counters and displaying them on framebuffer(currently 4224 points, generating 4224 vtx count and 4224*4 pixel count since each point is 2X2 pixels). Had to just slightly twist the test from the original test since after at context switch 64 pixels are given to output no matter how many are rendered (pixel vector). So the vertex data had to carry appropriate number of points so that we get a continous pixel count. Need to still put the register indexing inside the test

Change 128491 on 2003/10/24 by mangeshn@fl_mangeshn

added coissue tests for the pred_set_inv and pred_set_pop instructions

Change 128406 on 2003/10/24 by kevino@kevino_r400_win_marlboro

Fixed some of the test cases. Prim dimesnions exceeded display dimensions and were causing an rb assert. Not sure qute why, though. Maybe raster scissors need to be set up as the display dimensions.

Change 128395 on 2003/10/24 by mangeshn@fl_mangeshn

added coissue tests for pred_sete, pred_setne, pred_setgt and pred_setge instructions

Change 128392 on 2003/10/24 by kevino@kevino_r400_win_marlboro

Added several large odd textures with mipmapping to tp_multitexture_02 in tp_mt_tcfunc_2D_filter.h

Change 128356 on 2003/10/24 by kevino@kevino_r400_win_marlboro

Added getcomplod and getgradients testcases to tp_cubic.cpp

Change 128340 on 2003/10/24 by mkelly@fl_mkelly_r400_win_laptop

Moving and modifying pipe disable tests in sys/rom

Change 128338 on 2003/10/24 by cbrennan@cbrennan_r400_emu

Add stacks to tc's smoke.rg

Change 128333 on 2003/10/24 by mkelly@fl_mkelly_r400_win_laptop

Add handling for test_lib/src/chip/sys/rom

Change 128317 on 2003/10/24 by mkelly@fl_mkelly_r400_win_laptop

Changed stipple auto reset control to "1" instead of "2", since the BOOL data type
was removed from pa.blk.

Change 128283 on 2003/10/23 by ashishs@fl_ashishs_r400_win2

finally got the test working perfectly with help from Carlos. Now just need to have color same as the previous test

Change 128261 on 2003/10/23 by smoss@smoss_xenos_linux_orl

added rom, housekeeping

Change 128252 on 2003/10/23 by jhoule@jhoule_r400_linux_marlboro

Added ability to print input and output headers.
Cleaned up code to be more consistent across testbenches.

Change 128247 on 2003/10/23 by kevino@kevino_r400_win_marlboro

Adding mip packing setting to tp_multitexture_02. Defaulted to enabled.

Change 128213 on 2003/10/23 by mangeshn@fl_mangeshn

added coissue tests for sqrt_iee, fract, trunc and floor instructions

Change 128202 on 2003/10/23 by georgev@devel_georgev_r400_lin2_marlboro_tott

Added tests for non mod 32 texture pitches.

Change 128201 on 2003/10/23 by ashishs@fl_ashishs_r400_win2

correcting the test since the test has dounble the number of indices

Change 128193 on 2003/10/23 by ashishs@fl_ashishs_r400_win2

initial checkin for the test with vertex counter, pix counter and the vsisr_cont enabled

Change 128161 on 2003/10/23 by mkelly@fl_mkelly_r400_win_laptop

Change texture to work with Jocelyn's changes...

Change 128158 on 2003/10/23 by mkelly@fl_mkelly_r400_win_laptop

Change texture to work with Jocelyn's changes...

Change 128128 on 2003/10/23 by mkelly@fl_mkelly_r400_win_laptop

Add rts_wait_until

Change 128124 on 2003/10/23 by cbrennan@cbrennan_r400_release

Enabled endian swap and border color randomizations..

Change 128098 on 2003/10/23 by ashishs@fl_ashishs_r400_win

updated the descriptions and addded test in tracker

Change 128090 on 2003/10/23 by ashishs@fl_ashishs_r400_win2

Adding test to do vertex and pixel count in the same test. Thanks for the tip from Carlos the test works correctly

Change 128078 on 2003/10/23 by mkelly@fl_mkelly_r400_win_laptop

Move RT idle to after initiator...

Change 128067 on 2003/10/23 by mkelly@fl_mkelly_r400_win_laptop

Remove two tests from regressions until I fix them...

Change 128064 on 2003/10/23 by mkelly@fl_mkelly_r400_win_laptop

Moved WAIT_RTS_UNTIL.wait_rt_idle to be immediately after RTS initiator.

Change 128050 on 2003/10/23 by kevino@kevino_r400_win_marlboro

Fixed test-created sp file so that settexlod is right before the tfetches that use it. If ALu instr's are in between them.

Change 128047 on 2003/10/23 by georgev@devel_georgev_r400_lin2_marlboro_tott

Added debug register read on test completion.

Change 128041 on 2003/10/23 by kevino@kevino_r400_win_marlboro

Needed by tp_multitexture_01 and _02

Change 128031 on 2003/10/23 by ashishs@fl_ashishs_r400_win

due to s script problem we need to add new line at the end of the test_list

Change 128017 on 2003/10/23 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 128016 on 2003/10/23 by mkelly@fl_mkelly_r400_win_laptop

Adjustments...

Change 128015 on 2003/10/23 by mkelly@fl_mkelly_r400_win_laptop

Adjustments...

Change 128013 on 2003/10/23 by mkelly@fl_mkelly_r400_win_laptop

Update comments...

Change 128012 on 2003/10/23 by mkelly@fl_mkelly_r400_win_laptop

Set size correctly.

Change 127974 on 2003/10/22 by ashishs@fl_ashishs_r400_win2

Finally got the test working with help from Carlos and Laurent. Had test issues so just simplified the test to render as many points as needed without caring about the render packet size in one render pass

Change 127958 on 2003/10/22 by mangeshn@fl_mangeshn

added co-issue tests for recip_sqrt_clamped, recip_sqrt_ff and recip_sqrt_ieee. Updated SP test list

Change 127936 on 2003/10/22 by kevino@kevino_r400_win_marlboro

Commented out dumpPPM commands so that the tests wouldn't write out the textures to ppm files.

Change 127931 on 2003/10/22 by kevino@kevino_r400_win_marlboro

Commented out SaveBaseMap and SaveMipLevels where they were set to true- no need to dump them except for when writing the test and making sure the textures come out as expected.

Change 127911 on 2003/10/22 by mangeshn@fl_mangeshn

modified test data set and switch off #VERBOSE. Updated SP test list

Change 127901 on 2003/10/22 by mkelly@fl_mkelly_r400_win_laptop

More simple texture cases

Change 127891 on 2003/10/22 by tmartin@tmartin_r400_win

first complete version of the test pending a fix in the emulator

Change 127889 on 2003/10/22 by tmartin@tmartin_r400_win

cleaned up a couple areas of the code so they don't cause confusion in the future

Change 127887 on 2003/10/22 by domachi@diotargetxp

Add some output to debug random seed problem

Change 127868 on 2003/10/22 by kevino@kevino_r400_win_marlboro

Added getcomplod tests for 1D, 3D, and Cubic maps. Tex Stacks still need to be done.

Change 127867 on 2003/10/22 by mkelly@fl_mkelly_r400_win_laptop

More RTS texture samples...

Change 127803 on 2003/10/22 by mangeshn@fl_mangeshn

added coissue test for exp_ieee

Change 127801 on 2003/10/22 by mangeshn@fl_mangeshn

updated shaders

Change 127778 on 2003/10/22 by mkelly@fl_xenos_regspec

Update

Change 127770 on 2003/10/22 by mkelly@fl_mkelly_r400_win_laptop

Update for latest simd arch

Change 127757 on 2003/10/22 by mkelly@fl_mkelly_r400_win_laptop

VC address clamp disable by default.

Change 127756 on 2003/10/22 by mkelly@fl_mkelly_r400_win_laptop

VC Address clamp disable by default

Change 127751 on 2003/10/22 by mkelly@fl_mkelly_r400_win_laptop

Address clamp disable by default

Change 127740 on 2003/10/22 by mkelly@fl_mkelly_r400_win_laptop

Disable address clamping in VC

Change 127675 on 2003/10/21 by grayc@grayc_xenos_linux_orl

new script to generate a rg file

Change 127667 on 2003/10/21 by tmartin@tmartin_r400_win

removed some extraneous set fifo depth commands

Change 127663 on 2003/10/21 by tmartin@tmartin_r400_win

made some texture constant changes

Change 127654 on 2003/10/21 by mangeshn@fl_mangeshn

added co-issue tests for log_clamped and log_ieee

Change 127636 on 2003/10/21 by mkelly@fl_mkelly_r400_win_laptop

RECTANGLE_LIST, 4 textures, 64x64

Change 127631 on 2003/10/21 by mkelly@fl_mkelly_r400_win_laptop

nonRTS, 4 textures, 64x64, using to debug RTS 4 textures...

Change 127627 on 2003/10/21 by mangeshn@fl_mangeshn

added recip_ieee and recip_ff coissue tests

Change 127621 on 2003/10/21 by ashishs@fl_ashishs_r400_win

Added an option -g to goldenise the param tests in PERFORCE, but however this option doesnt check if the image PASSED. Kept this feature for later and just got the above functionality working

Change 127618 on 2003/10/21 by ashishs@fl_ashishs_r400_win

had some unknown problem

Change 127613 on 2003/10/21 by kevino@kevino_r400_win_marlboro

Added several 2D getcomplod test cases. Still need to add one for each of 1D, 3D, Cubic, and Tex Stack.

Change 127606 on 2003/10/21 by ashishs@fl_ashishs_r400_win

Goldenised the r400sc_multi* param tests for AMY after learning how to goldenise the param tests from Chris

Change 127558 on 2003/10/21 by mangeshn@fl_mangeshn

added test for - scalar recip_clamp inst tested with all vector inst for coissue

Change 127543 on 2003/10/21 by tien@tien_r400_devel_marlboro

Sanity check-in of TP block perf tests
Did not run release_parts_lib.pl, but as these tests
 are not in regression and are not RTL, I should be able to
 get away with it...
No other test should be using the 2 shader programs either!!!

Change 127536 on 2003/10/21 by ashishs@fl_ashishs_r400_win

Adding 3 new tests to test_list and Test_tracker.

Also updating description inside the tests. Also inside pix shader test increased the count from 512 to 16895 as earlier.

Change 127529 on 2003/10/21 by mkelly@fl_mkelly_r400_win_laptop

Corrected textures 0-3 sizes to match gold image after Chris Brennan's CL 125305.

Change 127505 on 2003/10/21 by domachi@diotargetxp

Ensure random shader generation is seeded with the -seed command line argument.

Change 127450 on 2003/10/20 by ashishs@fl_ashishs_r400_win2

Changed the test so that it uses the RENDER_ENGINE::INDIRECT_COMMAND_STREAM_MODE as recomemded by Kevin Ryan and Carlos so that these tests PASS in hardware.(currently FAILING in hardware due to high number of packets). Need to verify with Dan Clifton if the same PASS in hardware now ???

Change 127429 on 2003/10/20 by ashishs@fl_ashishs_r400_win2

Changing the test so that it uses the auto generated pixel shader counter to create an export address.

Change 127415 on 2003/10/20 by llefebvr@llefebvr_r400_emu_montreal

updated tp_simple_02 vertex shader to new VS 2.0 with serialize.

Change 127409 on 2003/10/20 by llefebvr@llefebvr_r400_emu_montreal

changed the x_mask test to use shader version 2 with serialize instead of the old v1.0

Change 127381 on 2003/10/20 by tmartin@tmartin_r400_win

added the vc test r400vc_data_format_01

Change 127380 on 2003/10/20 by mkelly@fl_mkelly_r400_win_laptop

Load constants in multiples of 4.

Change 127375 on 2003/10/20 by kevino@kevino_r400_win_marlboro

Some changes to tp_cubic.cpp to make sure color doesn't go to 0 when one of the tex sizes goes to 1.

Change 127370 on 2003/10/20 by ashishs@fl_ashishs_r400_win2

Editing the FAILING SP tests. These tests were FAILINg due to the predication optimization put by Laurent. Were valid tests earlier, but the emulator was changed to match hardware.

Change 127359 on 2003/10/20 by cbrennan@cbrennan_r400_emu

add cubic rg file.

Change 127352 on 2003/10/20 by mkelly@fl_mkelly_r400_win_laptop

Load constants in multiples of 4.

Change 127350 on 2003/10/20 by mkelly@fl_mkelly_r400_win_laptop

Display exported data in framebuffer for test debugging.

Change 127343 on 2003/10/20 by mkelly@fl_mkelly_r400_win_laptop

Load constants in multiples of 4.

Change 127340 on 2003/10/20 by mkelly@fl_mkelly_r400_win_laptop

Load constants in multiples of 4.

Change 127335 on 2003/10/20 by mkelly@fl_mkelly_r400_win_laptop

Load constants in multiples of 4.

Change 127331 on 2003/10/20 by tmartin@tmartin_r400_win

first gold check in

Change 127330 on 2003/10/20 by tmartin@tmartin_r400_win

first check in

Change 127327 on 2003/10/20 by tmartin@tmartin_r400_win

updated texture constants

Change 127176 on 2003/10/17 by kevino@kevino_r400_win_marlboro

Added large tex cases.

Change 127172 on 2003/10/17 by tmartin@tmartin_r400_win

added r400vc_cntl_07

Change 127167 on 2003/10/17 by tmartin@tmartin_r400_win

a slight twist on the cntl_04 test that stresses the coherency regs

Change 127163 on 2003/10/17 by tmartin@tmartin_r400_win

cleaned up the code

Change 127151 on 2003/10/17 by ashishs@fl_ashishs_r400_win2

editing the test so now it does the address export as well as the generated counter value for pix. Just to show that the address generated is correct and still the pix shader counter value is incorrect.

Change 127107 on 2003/10/17 by cbrennan@cbrennan_r400_emu

Aesthetic fix.

Change 127101 on 2003/10/17 by tmartin@tmartin_r400_win

added r400vc_rsp_01

Change 127097 on 2003/10/17 by tmartin@tmartin_r400_win

fixed typo

Change 127086 on 2003/10/17 by tmartin@tmartin_r400_win

one case was drawn overlapping another. this is fixed

Change 127049 on 2003/10/16 by tien@tien_r400_devel_marlboro

Updated perf tests

Change 127034 on 2003/10/16 by kevino@kevino_r400_win_marlboro

Updated tp_multitexture_02 randomized case

Change 127032 on 2003/10/16 by smoss@smoss_crayola_win

added vc

Change 127025 on 2003/10/16 by cbrennan@cbrennan_r400_emu

Saving changes to test that allow me to test to make sure mip faces are flipped properly.

Change 127005 on 2003/10/16 by georgev@devel_georgev_r400_lin2_marlboro_tott

Made tests perspective correct by default.

Change 127003 on 2003/10/16 by tmartin@tmartin_r400_win

test the redundant shader pipe functionality

Change 126998 on 2003/10/16 by tmartin@tmartin_r400_win

checkpoint

Change 126989 on 2003/10/16 by cbrennan@cbrennan_r400_win_branch

Fix shader program to new new new new new new new cube map instruction.

Change 126947 on 2003/10/16 by kevino@kevino_r400_win_marlboro

Got rid of old highlat test cases that were intended for back when the TC tried to let texture fetches slip past high-latency vtx requests

Change 126925 on 2003/10/16 by tmartin@tmartin_r400_win

added more fifo tests and fixed a calculation error in the spreadsheet

Change 126920 on 2003/10/16 by tmartin@tmartin_r400_win

delete

Change 126912 on 2003/10/16 by tmartin@tmartin_r400_win

added new tests to vary the l2 fifo depth

Change 126909 on 2003/10/16 by tmartin@tmartin_r400_win

split different fifo depth settings into separate files to work with the vc testbench

Change 126887 on 2003/10/16 by smoss@smoss_crayola_win

build test list true

Change 126877 on 2003/10/15 by csampayo@fl_csampayo3

Updated for the following tests:
r400sq_gpr_index_01
r400sq_gpr_index_02

Change 126864 on 2003/10/15 by tmartin@tmartin_r400_win

added a better description

Change 126858 on 2003/10/15 by cbrennan@cbrennan_r400_release

Cube mapping should clamp edge texels to get rid of seams. This is NOT the mip issue tho.

Change 126856 on 2003/10/15 by ashishs@fl_ashishs_r400_win2

Test to generate counter using gen_index_vtx and as well as using the 2 component
vsisr to fetch the data and validating it to be correct. Currently the counter is
set so that it can count till 16895 (basically 16895 points/vertices are rendered
in the program). But can be changed to increase and decrease the number of
points/vertices and thereby increase/decrease the auto counter generated by
gen_index_vtx. Also the counter has been exported to memory to see the value
that it is incremented by is correct. Also since the vtx shader uses dual component
feature of vsisr, the vertex data as well as indices has been appropriately
evaluated to cause a difference in the image for the same effect.

The counter can be increased or decraesed conveniently using the variable
in the test as follows :
const uint32 aperture = 16895 ; //atleast has to be 256 or RENDER PACKET SIZE

Change 126855 on 2003/10/15 by cbrennan@cbrennan_r400_release

Tweaks to cache thrash test to hopefully make it better.

Change 126836 on 2003/10/15 by tmartin@tmartin_r400_win

added the rest of the precision tests to the regression and removed the debug register from full chip

Change 126819 on 2003/10/15 by cbrennan@cbrennan_r400_release

Add a cache thrash test.
edit tests to turn off back face culling because half of the randoms werent displaying the image.
Add some rg files i want to keep around.
Add random tests for the cache thrashing and face crossing to the TC and TP suite.

Change 126809 on 2003/10/15 by kevino@kevino_r400_win_marlboro

Added non-pow2 and rectangular test cases. Also added random texture format tests. Made random tex sized non-pow2.

Change 126797 on 2003/10/15 by csampayo@fl_csampayo_r400

Adding GPR loop indexing tests

Change 126785 on 2003/10/15 by tmartin@tmartin_r400_win

set the texture constant pitch

Change 126759 on 2003/10/15 by kevino@kevino_r400_win_marlboro

Added w to tp_cubic.cpp. Added persp case which changes w from 1.0.

Change 126754 on 2003/10/15 by georgev@devel_georgev_r400_lin2_marlboro_tott

Even more aniso 14to1 fixes.

Change 126740 on 2003/10/15 by tmartin@tmartin_r400_win

enabled the second case

Change 126728 on 2003/10/15 by georgev@devel_georgev_r400_lin2_marlboro_tott

More changes for 14to1 aniso.

Change 126727 on 2003/10/15 by kevino@kevino_r400_win_marlboro

Added various random test cases:
randomized :   random stq,   ordered random xy
random_vtx_stq :   random stq,   original xy
random_vtx_xy_ordered : original stq, ordered random xy
random_vtx_xy :   original stq, random xy
random_vtx_all :   random stq,   random xy

ordered random xy means that it will draw a tri strip that does not fold back over itself.

Change 126694 on 2003/10/15 by cbrennan@cbrennan_r400_release

Renamed tp_mip_cubic to tp_mip_face_cross

Change 126646 on 2003/10/14 by kevino@kevino_r400_win_marlboro

Updated tp_cubic test that hits the faces correctly. Also, draw multiple rows, each downscalled from the last to hit the various mip levels. (mip level is encoded in green=level/16.0)

Change 126644 on 2003/10/14 by tmartin@tmartin_r400_win

updates

Change 126613 on 2003/10/14 by georgev@devel_georgev_r400_lin2_marlboro_tott

Changed texture map for 14to1 aniso debug.

Change 126561 on 2003/10/14 by ashishs@fl_ashishs_r400_win2

Test to validate the gen_index_pix counter. The test doesnt seem to produce the counter correctly, so currently under validation.

Change 126551 on 2003/10/14 by georgev@devel_georgev_r400_lin2_marlboro_tott

Reactivated border fraction and get gradients (new shaders).

Change 126546 on 2003/10/14 by kevino@kevino_r400_win_marlboro

1st cut at cubic test, but I think I am always getting just one face.

Change 126538 on 2003/10/14 by jhoule@jhoule_r400_linux_marlboro_reg

Temporarily adding a simpler version of GetBorderColorFraction for the RTL to test against

Change 126537 on 2003/10/14 by cbrennan@cbrennan_r400_release

Tweaked range of random coords.

Change 126523 on 2003/10/14 by domachi@diotargetxp

Fix vertex fetch stride of position shader when using the Random Shader Generator (RSG)

Change 126521 on 2003/10/14 by ashishs@fl_ashishs_r400_win2

Tests changed due to Chris Brenan's change #125305 Needed to shift the texture coord by 3 bits. Essentially "SIZE_1" in all the tests below have been shifted by 3 bits

Change 126460 on 2003/10/13 by ashishs@fl_ashishs_r400_win

correcting the test

Change 126436 on 2003/10/13 by kevino@kevino_r400_win_marlboro

Added new max/min mip clamp tests

Change 126406 on 2003/10/13 by ashishs@fl_ashishs_r400_win

optimizing

Change 126403 on 2003/10/13 by ashishs@fl_ashishs_r400_win

can safely increase the number of points to 16384 to be displayed on the screen

Change 126390 on 2003/10/13 by ashishs@fl_ashishs_r400_win

Adding test to gen_index_vtx

Change 126360 on 2003/10/13 by kevino@kevino_r400_win_marlboro

Set volume mag.min filter to be linear in the BLL cases.

Change 126355 on 2003/10/13 by kevino@kevino_r400_win_marlboro

Just added a few commented lines (which show various tfetch options of using comp and/or reg grad)

Change 126351 on 2003/10/13 by kevino@kevino_r400_win_marlboro

Fixed getset_gradient test to get the grads, square them or double them, and set back to H and V, then lookup texture with new grads.
Changed texture for grad_exp_adjust tests

Change 126348 on 2003/10/13 by tien@tien_r500_emu

Changed word counts for lod/coord FIFO to 32 to match RAM
Added ati_dff_in to tp4_tc testbench for TP_SQ_dec
Misc. changes to perf test...

Change 126346 on 2003/10/13 by kevino@kevino_r400_win_marlboro

Added grad_exp_adjust_h and _v testcases:
aniso_2D_grad_exp_adjust_neg_16_9
aniso_2D_grad_exp_adjust_neg_8_1
aniso_2D_grad_exp_adjust_pos0_7
aniso_2D_grad_exp_adjust_pos8_15

Change 126340 on 2003/10/13 by domachi@diotargetxp

Fix problem where multiple parameter cache allocs occured were generated using random shader generation. The position shader should not export to the parameter cache, but let the random shader generator do that.

Change 126339 on 2003/10/13 by kevino@kevino_r400_linux_marlboro

Changes pos testcase to have exp_adj_all of 31, which is max, not 32.

Change 126333 on 2003/10/13 by tmartin@tmartin_r400_win

added r400vc_fmt_precision_05 and r400vc_fmt_precision_06

Change 126328 on 2003/10/13 by cbrennan@cbrennan_r400_win_branch

increase random fmt range to hit 61

Change 126323 on 2003/10/13 by jhoule@jhoule_r400_linux_marlboro

Changed default test

Change 126321 on 2003/10/13 by ashishs@fl_ashishs_r400_win

setting up constants inside the test since was being used inside the pix shader

Change 126226 on 2003/10/10 by cbrennan@cbrennan_r400_emu

Release from my emu branch: texture stacks for TP as well.
Leda rule tweaks
add more .rg files

Change 126223 on 2003/10/10 by tmartin@tmartin_r400_win

stress test 1

Change 126194 on 2003/10/10 by ashishs@fl_ashishs_r400_win2

Changing the tests so that they output screen XY. The tests had PARAM_GEN = 1 or PARAM_GEN_RTS =1 but didnt have OUTPUT_SCREEN_XY = 0 or OUTPUT_SCREEN_XY_RTS = 0 because of which it caused problem in hardware, getting uninit data on OUTPUT_SCREEN_XY

Change 126090 on 2003/10/10 by ashishs@fl_ashishs_r400_win

Changing the shaders for the HOS adaptive tests so that while doing the mem exports the ea has MULADD instruction in it otherwise it asserts in SX

Change 126067 on 2003/10/10 by llefebvr@llefebvr_r400_emu_montreal

Swaping no flush for a real gfxIdle.

Change 126063 on 2003/10/10 by llefebvr@llefebvr_r400_emu_montreal

Put the wait in the wrong place.... It wasn't doing anything.

Change 126062 on 2003/10/10 by llefebvr@llefebvr_r400_emu_montreal

Inserting wait GFX idle to wait for results of pass 1 before going to 2.

Change 126025 on 2003/10/10 by cbrennan@cbrennan_r400_release

Fixed some test errors.

Change 125992 on 2003/10/09 by georgev@devel_georgev_r400_lin2_marlboro_tott

Added S and Q values of 0.5.

Change 125990 on 2003/10/09 by ashishs@fl_ashishs_r400_win

changing pred optimizations

Change 125987 on 2003/10/09 by ashishs@fl_ashishs_r400_win

changing pred optimization

Change 125983 on 2003/10/09 by ashishs@fl_ashishs_r400_win

removing pred optimizations

Change 125973 on 2003/10/09 by ashishs@fl_ashishs_r400_win2

correcting the path to the gold (t:\r400\gold -> t:\xenos\gold)

Change 125969 on 2003/10/09 by ashishs@fl_ashishs_r400_win

changing the shders for the predication optimiztion

Change 125965 on 2003/10/09 by omesh@omesh_r400_linux_marlboro

Added 2 and 4 sample specific versions of these tests on Larry's request. Verified that they compile and run on Linux.

Change 125957 on 2003/10/09 by llefebvr@llefebvr_r400_emu_montreal

Fixing RT test.

Change 125900 on 2003/10/09 by georgev@devel_georgev_r400_lin2_marlboro_tott

Partialy complete files. Do not use.

Change 125897 on 2003/10/09 by ashishs@fl_ashishs_r400_win2

changing the shaders to remove the optimizations for predications by manually putting (P) and (!P) and removing ALL IF's

Change 125805 on 2003/10/09 by lseiler@lseiler_r400_win_marlboro1

Removed 8-sample test from regress_rb

Change 125792 on 2003/10/09 by tmartin@tmartin_r400_win

converted vc test to tc test

Change 125730 on 2003/10/08 by ashishs@fl_ashishs_r400_win

Not use the optimization by setting manually (p) or (!p) before each ALU instruction (only the ones having address register, as recom by Laurent) and remove all ifs.

Change 125726 on 2003/10/08 by smoss@smoss_xenos_linux_orl

added vc stuff

Change 125721 on 2003/10/08 by ashishs@fl_ashishs_r400_win

Not use the optimization by setting manually (p) or (!p) before each ALU instruction (only the ones having address register, as recom by Laurent) and remove all ifs.

Change 125715 on 2003/10/08 by tmartin@tmartin_r400_win

changed the texture size field to reflect the new packing order

Change 125704 on 2003/10/08 by tmartin@tmartin_r400_win

commented out r400vc_base_addr_range_pci_02

Change 125696 on 2003/10/08 by georgev@devel_georgev_r400_lin2_marlboro_tott

Changed border color to white and changed wrap mode.

Change 125677 on 2003/10/08 by tmartin@tmartin_r400_win

added "MH.HDP_FB_START.write(frame_buffer_start);" to protect against bugs in future tests that use these as a base

Change 125671 on 2003/10/08 by ashishs@fl_ashishs_r400_win

Not use the optimization by setting manually (p) or (!p) before each ALU instruction (only the ones having address register, as recom by Laurent) and remove all ifs.

Change 125656 on 2003/10/08 by rmanapat@rmanapat_r400_release

Added til_fmt00_l and til_fmt01_l testcases

Change 125622 on 2003/10/08 by cbrennan@cbrennan_r400_emu

Integrate code from branch:
Implemented texture stacks in the TC.
integrated some rg files back from TOTT.
Tweaked leda rules for tca and tcb.
Added texture stacks to nightly tests and randoms.

Change 125613 on 2003/10/08 by lseiler@lseiler_r400_win_marlboro2

Increase precision of bypass path for 16-bit components, so that 16-bit RF can be generated accurately

Change 125588 on 2003/10/08 by georgev@devel_georgev_r400_lin2_marlboro_tott

Fix test bug with new FMT_1_1_1_1 format.

Change 125569 on 2003/10/08 by kevino@kevino_r400_linux_marlboro

rg file to run some of the dxt formats

Change 125555 on 2003/10/08 by tmartin@tmartin_r400_win

removed some unused code

Change 125510 on 2003/10/07 by tmartin@tmartin_r400_win

added r400vc_dword_alignment_01, r400vc_dword_alignment_02, r400vc_dword_alignment_03, r400vc_dword_alignment_04, r400vc_sector_reuse_01, r400vc_sector_reuse_02

Change 125484 on 2003/10/07 by georgev@devel_georgev_r400_lin2_marlboro_tott

Added some extra code for future tests.

Change 125468 on 2003/10/07 by georgev@devel_georgev_r400_lin2_marlboro_tott

First set of corrections.

Change 125455 on 2003/10/07 by tmartin@tmartin_r400_win

checks sector functionality and 512 bit request sizes

Change 125451 on 2003/10/07 by georgev@devel_georgev_r400_lin2_marlboro_tott

Changed 1 to 1 Aniso to Disable.

Change 125443 on 2003/10/07 by georgev@devel_georgev_r400_lin2_marlboro_tott

First round.

Change 125393 on 2003/10/07 by jhoule@jhoule_r400_linux_marlboro

Added pix_mask

Change 125367 on 2003/10/07 by mkelly@fl_mkelly_r400_win_laptop

Multi-pass pixel shader data dependent predication, 50% complete

Change 125358 on 2003/10/07 by tmartin@tmartin_r400_win

fixed a typo with one of the texture coordinate fetches

Change 125347 on 2003/10/07 by tmartin@tmartin_r400_win

test wrapping at the high end of the 32 bit address range

Change 125346 on 2003/10/07 by tmartin@tmartin_r400_win

test clamping at the high end of the 32 bit address range

Change 125341 on 2003/10/07 by tmartin@tmartin_r400_win

updated some non-essential code to make the test more complete

Change 125318 on 2003/10/07 by cbrennan@cbrennan_r400_emu

Reenable linear filtering.

Change 125305 on 2003/10/07 by cbrennan@cbrennan_r400_emu

Added fmt61 tests (U1111) and DXN tests that should have been there anyway.

Had x%2=0 and x%2=1 cases swapped for DXT3A_AS_1_1_1_1.
Also didnt have test cases defined for fmt49 in mip_cubic and stack tests.

Changed mip stacks to be multiples of 4 instead of powers of two.

Remove cp*e2 tests from other peoples sanity checks since TConst size field change breaks cp microcode.

Added FMT_DXT3A_AS_1_1_1_1 to emu, test lib, and tx_simple_* tests.

Separated 2d and stack size tconst packing, but left both at the same 13 bit fully packed.
Changed tconst size packing in HW
Changed testbench to turn on TPC checking more often. Was ignoring many fields when it thought they werent used.
Changed tcf_no_tpc in tc testbench to just be called tcf to keep waveform .rc compatibility with tp4_tc testbench.
Removed tests from emulator regress_e, release_parts_lib and daily_regress that failed with new size packing, but they are e2 tests which are no longer supported and need to have a microcode change to pass.

Added texture stacks to tests, primlib, cmn_lib, and emu.

Change 125302 on 2003/10/07 by tmartin@tmartin_r400_win

test the dword alignment with strides of 16 to 1 and move the base address so the stride reaches to the end of the cache line

Change 125295 on 2003/10/07 by cbrennan@cbrennan_r400_linux_marlboro

Last batch.

Change 125292 on 2003/10/07 by cbrennan@cbrennan_r400_linux_marlboro

rg file that is the release_parts_lib list.

Change 125290 on 2003/10/07 by cbrennan@cbrennan_r400_linux_marlboro

Added some rg files.

Change 125276 on 2003/10/07 by tmartin@tmartin_r400_win

initialized boolean constants before drawing points

Change 125270 on 2003/10/07 by mkelly@fl_mkelly_r400_win_laptop

Update for Xenos

Change 125262 on 2003/10/07 by tmartin@tmartin_r400_win

fixed a typo. reset the case loop start to 0

Change 125259 on 2003/10/07 by mkelly@fl_mkelly_r400_win_laptop

Change for Xenos msaa

Change 125255 on 2003/10/07 by mkelly@fl_mkelly_r400_win_laptop

Change msaa 8 to msaa 4 for Xenos

Change 125174 on 2003/10/06 by tmartin@tmartin_r400_win

the final dword of memory is now getting accessed

Change 125170 on 2003/10/06 by tmartin@tmartin_r400_win

fixed some typos

Change 125161 on 2003/10/06 by georgev@devel_georgev_r400_lin2_marlboro_tott

Changed interlaced format to 4x4.

Change 125131 on 2003/10/06 by smoss@smoss_xenos_linux_orl

increased time between bsubs

Change 125057 on 2003/10/06 by kevino@kevino_r400_emu

Added this in so it can pick up GetEnvPath from gfx_utils

Change 125045 on 2003/10/06 by ashishs@fl_ashishs_r400_win

Adding Chris Hammer to the nightly regressions

Change 125024 on 2003/10/06 by ashishs@fl_ashishs_r400_win

Added Mark to nightly regressions so he could get the SC results

Change 124946 on 2003/10/03 by cbrennan@cbrennan_r400_release

Forgot to release test change to reduce known broken cases for the time being.

Change 124916 on 2003/10/03 by mangeshn@fl_mangeshn

added dot2add inf/nan test to the test list

Change 124915 on 2003/10/03 by ashishs@fl_ashishs_r400_win2

making a cp_regress_group and adding John, Alex, Frank and Mark to it

Change 124912 on 2003/10/03 by ashishs@fl_ashishs_r400_win2

Adding Mark Earl to the nightly regressions

Change 124911 on 2003/10/03 by mkelly@fl_mkelly_r400_win_laptop

Set num samples to 4msaa

Change 124909 on 2003/10/03 by mangeshn@fl_mangeshn

upadting tests

Change 124902 on 2003/10/03 by tmartin@tmartin_r400_win

added r400vc_fetch_addr_range_03

Change 124870 on 2003/10/03 by ashishs@fl_ashishs_r400_win

updating the path for the trackers which were earlier under t:\r400\ and now under t:\xenos\

Change 124868 on 2003/10/03 by georgev@devel_georgev_r400_lin2_marlboro_tott

Removed FMT_32_32_32_FLOAT from test list.

Change 124859 on 2003/10/03 by mkelly@fl_mkelly_r400_win_laptop

Real Time Stream with max nested subroutines and loops with 4 parameter dependent predication.

Change 124851 on 2003/10/03 by llefebvr@llefebvr_r400_emu_montreal

Interpolation precision change to meet HW and timing.

Change 124830 on 2003/10/03 by mkelly@fl_mkelly_r400_win_laptop

remove r400sc_template_01

Change 124817 on 2003/10/03 by mangeshn@fl_mangeshn

update

Change 124815 on 2003/10/03 by mangeshn@fl_mangeshn

update

Change 124814 on 2003/10/03 by mangeshn@fl_mangeshn

update

Change 124808 on 2003/10/03 by kevino@kevino_r400_linux_marlboro

P4 delete of old outdated aniso test (that didn't do aniso correctly anyway)

Change 124798 on 2003/10/03 by mangeshn@fl_mangeshn

updated DOT4 inf/nan

Change 124794 on 2003/10/03 by ashishs@fl_ashishs_r400_win2

sorting test_list

Change 124793 on 2003/10/03 by mangeshn@fl_mangeshn

updated DOT3 inf/nan to new template

Change 124789 on 2003/10/03 by ashishs@fl_ashishs_r400_win2

adding tests

Change 124783 on 2003/10/03 by ashishs@fl_ashishs_r400_win2

adding floor scalr instruction

Change 124777 on 2003/10/03 by ashishs@fl_ashishs_r400_win2

added test for vector floor

Change 124767 on 2003/10/03 by ashishs@fl_ashishs_r400_win2

Added test for floor vector

Change 124765 on 2003/10/03 by ashishs@fl_ashishs_r400_win2

editing the test to make it scalar only test

Change 124764 on 2003/10/03 by tmartin@tmartin_r400_win

fixed some bugs

Change 124762 on 2003/10/03 by kevino@kevino_r400_linux_marlboro

Fixed display pitch- had not set it to match disp_x_dim for aniso cases

Change 124759 on 2003/10/03 by ashishs@fl_ashishs_r400_win2

disabling print statements and cleaning up

Change 124753 on 2003/10/03 by ashishs@fl_ashishs_r400_win2

Added trunc vector operation

Change 124728 on 2003/10/03 by ashishs@fl_ashishs_r400_win2

inf nan test for scalar trunc

Change 124727 on 2003/10/03 by mangeshn@fl_mangeshn

updated to final version

Change 124725 on 2003/10/02 by ashishs@fl_ashishs_r400_win2

adding vector instruction

Change 124724 on 2003/10/02 by ashishs@fl_ashishs_r400_win2

seperating out scalar and vector instruction, submit scalar

Change 124722 on 2003/10/02 by ashishs@fl_ashishs_r400_win2

finalising the shader and removing scalar instruction specific stuff

Change 124691 on 2003/10/02 by kevino@kevino_r400_linux_marlboro

Fixed tp_multitexture_01 and _02 aniso tests so that they do the levels they say they do.

Change 124677 on 2003/10/02 by mangeshn@fl_mangeshn

preliminary check in for dot2add inf/nan

Change 124672 on 2003/10/02 by tmartin@tmartin_r400_win

added
- r400vc_fmt_precision_08
- r400vc_fmt_precision_10
- r400vc_fetch_addr_range_02

Change 124670 on 2003/10/02 by tmartin@tmartin_r400_win

test when a negative offset causes the fetch address to be less than 0

Change 124669 on 2003/10/02 by ashishs@fl_ashishs_r400_win2

adding 'VfUseTc = 0' instead of 'UseVC = 1'

Change 124655 on 2003/10/02 by ashishs@fl_ashishs_r400_win2

Adding the sin-cos tests

Change 124654 on 2003/10/02 by kevino@kevino_r400_linux_marlboro

Update the aniso tests to hit the advertised ratios. The *128x128* testcases set max aniso to 16:1 no matter what and rely on the prim setup to hit the correct ratios.

Change 124628 on 2003/10/02 by cbrennan@cbrennan_r400_linux_marlboro

Cripple test temporarily from doing things that we know are broken.

Change 124598 on 2003/10/02 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint on RTS parameter dependent predication

Change 124579 on 2003/10/02 by ashishs@fl_ashishs_r400_win2

initial checkin for test to do fract vector operation

Change 124545 on 2003/10/02 by tmartin@tmartin_r400_win

32 fixed precision tests

Change 124542 on 2003/10/02 by georgev@devel_georgev_r400_lin2_marlboro_coverage

First revision. Private use only.

Change 124527 on 2003/10/02 by ashishs@fl_ashishs_r400_win

Adding Mantor and Randy to the list

Change 124523 on 2003/10/02 by kevino@kevino_r400_emu

Added "USE TC_FOR_VERTEX_FETCHES;" to remaining shader programs for tc_vfetch

Change 124519 on 2003/10/02 by ashishs@fl_ashishs_r400_win2

Changing the test to do just scalar operation (prev was doing vector too). Using export to memory and also generating a compare image

Change 124516 on 2003/10/02 by kevino@kevino_r400_linux_marlboro

Added rg/tc_vfetch.rg which runs all the test cases in this test.

Change 124436 on 2003/10/01 by ashishs@fl_ashishs_r400_win2

Adding test for SIN. Currently has some problems which will need to be taken care of.

Change 124430 on 2003/10/01 by tmartin@tmartin_r400_win

added r400vc_fmt_precision_int_09, r400vc_fmt_precision_09, r400vc_fmt_precision_int_08

Change 124412 on 2003/10/01 by mangeshn@fl_mangeshn

update

Change 124401 on 2003/10/01 by gregm@fl_gregm

update

Change 124388 on 2003/10/01 by kevino@kevino_r400_linux_marlboro

Added in 3d aniso test cases (4/format)

Change 124370 on 2003/10/01 by ashishs@fl_ashishs_r400_win2

Adding test for INF-NAN for COS

Change 124353 on 2003/10/01 by mangeshn@fl_mangeshn

split all scalar and vector combination tests to separate ones. Updated test list and test tracket to reflect new tests added.

Change 124332 on 2003/10/01 by tmartin@tmartin_r400_win

changed to test when the vertex fetch size constant is 0 and clamping is enabled. Previously it was thought that setting the size to 0 disabled clampling.

Change 124300 on 2003/10/01 by mkelly@fl_mkelly_r400_win_laptop

Add some comments...

Change 124299 on 2003/10/01 by kevino@kevino_r400_linux_marlboro

Changed vtx2 to have s,t =1.0, 1.0 (under george's direction) so the aniso test now gets to 16:1

Change 124297 on 2003/10/01 by mkelly@fl_mkelly_r400_win_laptop

data dep pred in Viz Query

Change 124269 on 2003/10/01 by kevino@kevino_r400_linux_marlboro

A large change to tp_multitexture_02. The testcases are now generated in a slightly different way, using a common tclist file instead of individual ones for each format. The format commp is cycled through USBG SBGU BGUS GUSB for the 4 textures in all cases. A tfc.validateGamma() call is used to change G to U if the format is not degammable.

Updated the tc_weekly file to run formats 7, 10, 11, 12. Updated tp4_tc_weekly.rg files to run ALL good formats (0, 1, 43-48 not run since they fail with emu errors.) This adds ~9000 cases to tp4_tc_weekly.rg

Finished off some integer and signed-rf mode tests.

Change 124267 on 2003/10/01 by mangeshn@fl_mangeshn

updating tests for framebuffer dump capability

Change 124266 on 2003/10/01 by ashishs@fl_ashishs_r400_win2

PRECISION test for COS

Change 124246 on 2003/10/01 by tmartin@tmartin_r400_win

Tests the precision of FMT_32_32_32_32_FLOAT with an unsigned integer frame buffer.

Change 124244 on 2003/10/01 by tmartin@tmartin_r400_win

updated to test a finer precision

Change 124231 on 2003/10/01 by mzini@mzini_crayola_linux_orl

No longer look at UseVc. VfUseTc in place

Change 124226 on 2003/10/01 by mkelly@fl_mkelly_r400_win_laptop

Update comments

Change 124217 on 2003/10/01 by mkelly@fl_mkelly_r400_win_laptop

Multi-pass pixel shading, data dep predication on pixel indice for memory export with data dep
pred on color import on second pixel pass.

Change 124207 on 2003/10/01 by tmartin@tmartin_r400_win

uncommented points

Change 124205 on 2003/10/01 by tmartin@tmartin_r400_win

added r400vc_fmt_precision_int_07

Change 124140 on 2003/09/30 by mangeshn@fl_mangeshn

adding missing test

Change 124137 on 2003/09/30 by tmartin@tmartin_r400_win

test the precision of the 16_16_16_16_FLOAT format

Change 124136 on 2003/09/30 by ashishs@fl_ashishs_r400_win2

test for SIN precision. Found to be very low precision (almost matching just 1 mantissa bit, but for very low numbers)

Change 124135 on 2003/09/30 by tmartin@tmartin_r400_win

added HDP_FB_START and make some other changes to update the test formatting

Change 124121 on 2003/09/30 by mangeshn@fl_mangeshn

updated tests for framebuffer dumps

Change 124115 on 2003/09/30 by jhoule@jhoule_r400_linux_marlboro

Adding endian swap cases

Change 124114 on 2003/09/30 by jhoule@jhoule_r400_linux_marlboro

Added support for ENDIAN_SWAP

Change 124104 on 2003/09/30 by mangeshn@fl_mangeshn

updated tests to include dumps to file for additional validation (switched off in the default case)

Change 124079 on 2003/09/30 by mkelly@fl_mkelly_r400_win_laptop

pred_setge_push w default check w/data dep pred secondary data fetch

Change 124054 on 2003/09/30 by mkelly@fl_mkelly_r400_win_laptop

pred_setgt_push default w check w/data dep pred and secondary data fetch

Change 124045 on 2003/09/30 by mkelly@fl_mkelly_r400_win_laptop

Update test comments

Change 124040 on 2003/09/30 by mkelly@fl_mkelly_r400_win_laptop

pred_setne_push default w check with data dep pred fetching

Change 124027 on 2003/09/30 by tmartin@tmartin_r400_win

set HDP_FB_START so the test doesn't hang

Change 124016 on 2003/09/30 by mkelly@fl_mkelly_r400_win_laptop

pred_sete_push default mode w/data dep pred and secondary fetching

Change 123999 on 2003/09/30 by mkelly@fl_mkelly_r400_win_laptop

pred_set_inv on W, data dependent secondary fetching...

Change 123961 on 2003/09/30 by gregm@fl_gregm

max min and muladd tests

Change 123866 on 2003/09/29 by mangeshn@fl_mangeshn

added missing test to SP test_list - thanks for pointing it out Ashish

Change 123855 on 2003/09/29 by ashishs@fl_ashishs_r400_win2

Adding 2 missing tests

Change 123849 on 2003/09/29 by tmartin@tmartin_r400_win

Makes sure that every select can choose every performance register type

Change 123838 on 2003/09/29 by mkelly@fl_mkelly_r400_win_laptop

pred_set_inv primary vertex buffer data dependent memory fetching
from a secondary buffer, cycling all fetch formats.

Change 123837 on 2003/09/29 by mangeshn@fl_mangeshn

updating test tracker and sp test list (inserting new tests according to new alpabhetical sorting)

Change 123829 on 2003/09/29 by ashishs@fl_ashishs_r400_win2

sorting test_list

Change 123827 on 2003/09/29 by mangeshn@fl_mangeshn

added tests for all the PRED_* instructions and inf/nan tests for the CND_* instructions

Change 123824 on 2003/09/29 by jhoule@jhoule_r400_linux_marlboro

Adding 2 simple aniso test files

Change 123784 on 2003/09/29 by tmartin@tmartin_r400_win

updated the description

Change 123772 on 2003/09/29 by tmartin@tmartin_r400_win

Tests the precision of FMT_32_32_32_32 with unsigned integers

Change 123766 on 2003/09/29 by kevino@kevino_r400_linux_marlboro

Decreased from maz tex size to max tez size -1. largetex_05 now hangs in sim, but not sure why.

Change 123751 on 2003/09/29 by ashishs@fl_ashishs_r400_win

Deleting test r400sc_point_jss_3X4_01

Change 123628 on 2003/09/26 by ctaylor@ctaylor_xenos_linux_orl

Change msaa num_samples to limit to 1,2,4 sample for Xenos and disable Rand Shaders for now.

Change 123616 on 2003/09/26 by mangeshn@fl_mangeshn

precision and nan/inf test fro PRED_SETE_PUSH

Change 123606 on 2003/09/26 by tmartin@tmartin_r400_win

test precision of 32_32_32_32_float

Change 123602 on 2003/09/26 by mangeshn@fl_mangeshn

precision and nan/inf test for PRED_SET_CLR

Change 123601 on 2003/09/26 by mangeshn@fl_mangeshn

precision and nan/inf test for PRED_SET_RESTORE

Change 123566 on 2003/09/26 by mangeshn@fl_mangeshn

added precision and nan/inf tests for: PRED_SET_INV and PRED_SET_POP

Change 123543 on 2003/09/26 by eberger@eberger_r400_linux_marlboro

Created two new tests to verify address calculation with a
variety of pitch values.

Change 123521 on 2003/09/26 by ashishs@fl_ashishs_r400_win2

disabling debug prints

Change 123516 on 2003/09/26 by mkelly@fl_mkelly_r400_win_laptop

Finalizing transition from 8 msaa to 4 for Xenos

Change 123514 on 2003/09/26 by ashishs@fl_ashishs_r400_win2

test for SUB_CONST opcode

Change 123510 on 2003/09/26 by mangeshn@fl_mangeshn

added precision and nan/inf tests for: PRED_SETE, PRED_SETNE, PRED_SETGT, PRED_SETGE

Change 123509 on 2003/09/26 by mkelly@fl_mkelly_r400_win_laptop

Update from 8 to 4 msaa disabled, Xenos

Change 123504 on 2003/09/26 by domachi@diotargetxp

Fixed problem where shaders for tc_random test only fetched from 2 textures. We now fetch from up to 4 seperate textures if specified by the test case.

Change 123501 on 2003/09/26 by mkelly@fl_mkelly_r400_win_laptop

msaa changes for Xenos

Change 123496 on 2003/09/26 by kevino@kevino_r400_linux_marlboro

Needed for tp_multitexture_02_perf

Change 123495 on 2003/09/26 by kevino@kevino_r400_linux_marlboro

Added a perf case that does 2 textures as dxt1 and 8888. The waves for this case need to be hand-checked to verify performance (and were before checkin)

Change 123494 on 2003/09/26 by mkelly@fl_mkelly_r400_win_laptop

Rename
//depot/r400/devel/test_lib/src/chip/gfx/sc/r400sc_line_msaa_8_textured_01_pix.sp To
//depot/r400/devel/test_lib/src/chip/gfx/sc/r400sc_line_msaa_4_textured_01_pix.sp

Change 123490 on 2003/09/26 by eberger@eberger_r400_linux_marlboro

Added several new tests to verify pixel formats when the
interpolated color is exported without clamping. Changed the
test r400rb_max_pitch.cpp to activate subtests that enable
multisampling with a depth buffer.

Change 123488 on 2003/09/26 by mkelly@fl_mkelly_r400_win_laptop

4 msaa changes for Xenos

Change 123487 on 2003/09/26 by mkelly@fl_mkelly_r400_win_laptop

8 msaa to 4 for Xenos

Change 123486 on 2003/09/26 by mkelly@fl_mkelly_r400_win_laptop

8 msaa to 4 for Xenos

Change 123484 on 2003/09/26 by mkelly@fl_mkelly_r400_win_laptop

del and rename test

Change 123482 on 2003/09/26 by mkelly@fl_mkelly_r400_win_laptop

Enable 4 msaa for Xenos

Change 123459 on 2003/09/26 by kevino@kevino_r400_linux_marlboro

Added ifdef for fmt_independent cases so they don't show up everywhere.

Change 123454 on 2003/09/26 by mkelly@fl_mkelly_r400_win_laptop

Temporarily comment out SIMD testing.

Change 123444 on 2003/09/26 by mkelly@fl_mkelly_r400_win_laptop

del some tests
update test_list

Change 123442 on 2003/09/26 by mkelly@fl_mkelly_r400_win_laptop

8 to 4 msaa for Xenos...

Change 123435 on 2003/09/26 by mkelly@fl_mkelly_r400_win_laptop

Add recip for new SPI requirements on prim type detection

Change 123432 on 2003/09/26 by mkelly@fl_mkelly_r400_win_laptop

update disp size on simple line list
del old test

Change 123428 on 2003/09/26 by mkelly@fl_mkelly_r400_win_laptop

fix

Change 123381 on 2003/09/25 by ashishs@fl_ashishs_r400_win

Adding 5 more tests

Change 123371 on 2003/09/25 by ashishs@fl_ashishs_r400_win

Adding test for MUL_CONST

Change 123361 on 2003/09/25 by brianf@brianf_r400_linux_marlboro

Removed from crayola

Change 123357 on 2003/09/25 by ashishs@fl_ashishs_r400_win

Added test for ADD_CONST

Change 123355 on 2003/09/25 by brianf@brianf_r400_linux_marlboro

Updated include path

Change 123352 on 2003/09/25 by tmartin@tmartin_r400_win

added some fmt_precision tests

Change 123340 on 2003/09/25 by mkelly@fl_mkelly_r400_win_laptop

Modify shaders for new Xenos SPI on primtype detection

Change 123336 on 2003/09/25 by brianf@brianf_r400_linux_marlboro

Moved az_tst library tests

Change 123335 on 2003/09/25 by eberger@eberger_r400_linux_marlboro

Initial version of a test to verify the memory export paths.

Change 123334 on 2003/09/25 by tmartin@tmartin_r400_win

added SERIALIZE instructions

Change 123326 on 2003/09/25 by ashishs@fl_ashishs_r400_win

PRECISION test for SUB_CONST opcode

Change 123323 on 2003/09/25 by mkelly@fl_mkelly_r400_win_laptop

8 to 4 msaa for Xenos

Change 123320 on 2003/09/25 by ashishs@fl_ashishs_r400_win

PRECISION test for MUL_COSNT opcode

Change 123316 on 2003/09/25 by ashishs@fl_ashishs_r400_win

PRECISION test for ADD_CONST opcode

Change 123313 on 2003/09/25 by mkelly@fl_mkelly_r400_win_laptop

8 to 4 msaa for Xenos

Change 123309 on 2003/09/25 by mkelly@fl_mkelly_r400_win_laptop

8 to 4 MSAA for xenos

Change 123306 on 2003/09/25 by tmartin@tmartin_r400_win

test precision in unsigned integer mode

Change 123305 on 2003/09/25 by tmartin@tmartin_r400_win

test precision of FMT_16_16_16_16

Change 123304 on 2003/09/25 by tmartin@tmartin_r400_win

updated to draw all intended points

Change 123277 on 2003/09/25 by mkelly@fl_mkelly_r400_win_laptop

8 to 4 msaa for xenos

Change 123273 on 2003/09/25 by mkelly@fl_mkelly_r400_win_laptop

8 to 4 msaa for xenos

Change 123267 on 2003/09/25 by mkelly@fl_mkelly_r400_win_laptop

8 -> 4 msaa for xenos

Change 123261 on 2003/09/25 by mkelly@fl_mkelly_r400_win_laptop

Change to 4 msaa for xenos

Change 123251 on 2003/09/25 by kevino@kevino_r400_linux_marlboro

Tp_multitexture_02 broken into tests with 5 formats each (10 doesn't compile on windows)
Updated tc_nightly and weekly to addd new test cases
Added fmt_independent test cases

Change 123239 on 2003/09/25 by mkelly@fl_mkelly_r400_win_laptop

Xenos change to 4 msaa

Change 123232 on 2003/09/25 by mkelly@fl_mkelly_r400_win_laptop

del, renamed to r400sc_xenos* and modified...

Change 123230 on 2003/09/25 by mkelly@fl_mkelly_r400_win_laptop

Update msaa for xenos

Change 123229 on 2003/09/25 by kevino@kevino_r400_linux_marlboro

tp_mutlitexture_02 testcase with 5 formats instead of 10. (10 still fails on windows)

Change 123221 on 2003/09/25 by kevino@kevino_r400_linux_marlboro

One forgotten file fromlast submit

Change 123220 on 2003/09/25 by kevino@kevino_r400_linux_marlboro

Added 3D texoffset and texsize tetscases
Reworked tp_multitexture_02 to be able to split it into multiple tests (since the 10K testcases causes Windows to report too many exceptions.) The "real" tp_multitexture_02 test code is in tp_multitexture_02_basecode.cpp. This (can be but) shouldn't be run by itself as a test. The other tp_multitexture_02*cpp tests define which formats they want to cover then include this basecode.
Note that the versions I checked in cover 10 formats each.
I also added a format06 -only test which compiles a lot faster, which is useful for testcase development.

Change 123206 on 2003/09/25 by mkelly@fl_mkelly_r400_win_laptop

8 MSAA -> 4 MSAA change

Change 123195 on 2003/09/25 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 123157 on 2003/09/24 by gregm@fl_gregm

submit

Change 123155 on 2003/09/24 by gregm@fl_gregm

infNan & precision mul_prev2

Change 123119 on 2003/09/24 by ashishs@fl_ashishs_r400_win

updated

Change 123102 on 2003/09/24 by mangeshn@fl_mangeshn

added tests fro nan/inf data for vector adn scalar forms for the following: KILLGE, KILLGT, KILLNE and KILLONE

Change 123099 on 2003/09/24 by ashishs@fl_ashishs_r400_win2

changed the script gold path to t:\xenos\gold\

Change 123096 on 2003/09/24 by mkelly@fl_mkelly_r400_win_laptop

xenos update...

Change 123095 on 2003/09/24 by mkelly@fl_mkelly_r400_win_laptop

xenos update

Change 123060 on 2003/09/24 by georgev@devel_georgev_r400_lin2_marlboro_tott

Added truncate and round tests.

Change 123045 on 2003/09/24 by mkelly@fl_mkelly_r400_win_laptop

Rename

Change 123033 on 2003/09/24 by mkelly@fl_mkelly_r400_win_laptop

Rename
//depot/r400/devel/test_lib/src/chip/gfx/sc/r400sc_msaa_8_simple_triangle_01.cpp To
//depot/r400/devel/test_lib/src/chip/gfx/sc/r400sc_msaa_4_simple_triangle_01.cpp

Change 123032 on 2003/09/24 by mkelly@fl_mkelly_r400_win_laptop

del for xenos

Change 123029 on 2003/09/24 by mkelly@fl_mkelly_r400_win_laptop

xenos msaa

Change 123026 on 2003/09/24 by mkelly@fl_mkelly_r400_win_laptop

Rename
//depot/r400/devel/test_lib/src/chip/gfx/sc/r400sc_line_expand_width_msaa_8_0*.cpp To
//depot/r400/devel/test_lib/src/chip/gfx/sc/r400sc_line_expand_width_msaa_4_0*.cpp

Change 123020 on 2003/09/24 by mkelly@fl_mkelly_r400_win_laptop

del non xenos

Change 123018 on 2003/09/24 by mkelly@fl_mkelly_r400_win_laptop

Update msaa for xenos

Change 123009 on 2003/09/24 by eberger@eberger_r400_linux_marlboro

Initial version of a test to verify that the zplane value is
different from the Z value that gets exported from the shader.

Change 122996 on 2003/09/24 by mkelly@fl_mkelly_r400_win_laptop

Add gold

Change 122984 on 2003/09/24 by kevino@kevino_r400_linux_marlboro

Added 3D mip and texture address offset cases. Currently available for fmt06 only.

Change 122973 on 2003/09/24 by mkelly@fl_mkelly_r400_win_laptop

del (backed up in sc_r500)

Change 122972 on 2003/09/24 by mkelly@fl_mkelly_r400_win_laptop

Need to back up these tests for any possibility of R500, since tests are starting to change for Xenos.

Change 122957 on 2003/09/24 by mkelly@fl_mkelly_r400_win_laptop

del

Change 122953 on 2003/09/24 by mkelly@fl_mkelly_r400_win_laptop

1. regress_e tests added to Makefile for 2 and 4 MSAA

2. changed num_samples in r400sc_poly_offset_05 to 0
3. added 2 and 4 MSAA tests to golden.lst

Change 122948 on 2003/09/24 by mkelly@fl_mkelly_r400_win_laptop

Remove immediately recogizable tests that are > 4 MSAA and will not be converted for xenos.

Change 122938 on 2003/09/24 by ashishs@fl_ashishs_r400_win2

completed with dumping expected image and also changing VTE settings to offset the normal data at a different place

Change 122937 on 2003/09/24 by ashishs@fl_ashishs_r400_win2

changing the data set to show that the overflow in shader causes clamp to MAX_FLOAT wheras on compiler causes to INF

Change 122936 on 2003/09/23 by ashishs@fl_ashishs_r400_win2

created a simple bug case for checking combinations of hex values on the input

Change 122933 on 2003/09/23 by mangeshn@fl_mangeshn

Added tests: INF/NAN for: SETE(scalar and vector), SETGE(scalar and vector), SETGT(scalar and vector), SETNE(scalar and vector), KILLE saclar and KILLE vector

Change 122907 on 2003/09/23 by tmartin@tmartin_r400_win

added more endian swap tests

Change 122905 on 2003/09/23 by tmartin@tmartin_r400_win

test endian swap with pci and agp

Change 122898 on 2003/09/23 by ctaylor@fl_ctaylor_r400_dtwin_marlboro

Removed 3,6,8 sample MSAA for Xenos. Removed 8-sample regression tests.

Change 122895 on 2003/09/23 by mangeshn@fl_mangeshn

Added the source code, updated SP test_list and the test_tracker for the following tests: Precision MOVA (both scalar and vector), Inf/Nan MOVA (scalar and vector), Precision Mova_Floor and Inf/Nan Mova_Floor.

Change 122887 on 2003/09/23 by smoss@smoss_xenos_linux_orl

comment out full chip only test

Change 122826 on 2003/09/23 by mzini@mzini_r400_win

Removed useTC flag

Change 122813 on 2003/09/23 by kevino@kevino_r400_linux_marlboro

Had mislabeled fmts 57-59 (format number didn't match format name). Fixed this.

Change 122804 on 2003/09/23 by mkelly@fl_mkelly_r400_win_laptop

New golds for 2/4 MSAA, Xenos...

Change 122802 on 2003/09/23 by mkelly@fl_mkelly_r400_win_laptop

del

Change 122797 on 2003/09/23 by kevino@kevino_r400_linux_marlboro

Added some useful .rg files to the rg subdirectory
Added tests to move mip address and added to tc_nightly.rg
Added tex offset cases from -8 to -4 and 4 to 7.5
Added unfinished 3d mip cases

Change 122795 on 2003/09/23 by georgev@devel_georgev_r400_lin2_marlboro_tott

Shortened max address tests so that they don't blow out buffers.
This did not work for sx_advanced_test.

Change 122770 on 2003/09/23 by mkelly@fl_mkelly_r400_win_laptop

fix sp name

Change 122761 on 2003/09/23 by mkelly@fl_mkelly_r400_win_laptop

rename

Change 122760 on 2003/09/23 by mkelly@fl_mkelly_r400_win_laptop

New test for xenos

Change 122754 on 2003/09/23 by mkelly@fl_mkelly_r400_win_laptop

New xenos test

Change 122719 on 2003/09/23 by mkelly@fl_mkelly_r400_win_laptop

New xenos test

Change 122712 on 2003/09/23 by mzini@mzini_r400_win

Changed test to force it to use the Texture Cache by default

Change 122707 on 2003/09/23 by mkelly@fl_mkelly_r400_win_laptop

Xenos specific 4 MSAA related test...

Change 122705 on 2003/09/23 by mkelly@fl_mkelly_r400_win_laptop

New test for 4 msaa

Change 122695 on 2003/09/23 by jhoule@jhoule_r400_win_lt

Simple test to verify minimal functionality of opcodes

Change 122688 on 2003/09/23 by kevino@kevino_r400_linux_marlboro

Both textures 0 and 1 were set to none. Set 0 back to an endian swap.

Change 122685 on 2003/09/23 by mkelly@fl_mkelly_r400_win_laptop

Change from 8 Msaa to 4 Msaa

Change 122682 on 2003/09/23 by mkelly@fl_mkelly_r400_win_laptop

4 MSAA CENTERS, CENTROIDS, CENTERS_AND_CENTROIDS

Change 122656 on 2003/09/22 by georgev@devel_georgev_r400_lin2_marlboro_tott

Added mask tests for pixel shaders.

Change 122654 on 2003/09/22 by gregm@fl_gregm

submit

Change 122652 on 2003/09/22 by gregm@fl_gregm

mul_prev precision and infNan tests

Change 122646 on 2003/09/22 by georgev@devel_georgev_r400_lin2_marlboro_tott

Add test names to previous update.

Change 122643 on 2003/09/22 by gregm@fl_gregm

submit

Change 122639 on 2003/09/22 by gregm@fl_gregm

update

Change 122637 on 2003/09/22 by gregm@fl_gregm

mul tests

Change 122636 on 2003/09/22 by ashishs@fl_ashishs_r400_win

initial checkin for infNan test for CUBE

Change 122635 on 2003/09/22 by tmartin@tmartin_r400_win

added r400vc_debug_01

Change 122634 on 2003/09/22 by georgev@devel_georgev_r400_lin2_marlboro_tott

Odd size texture maps added.

Change 122633 on 2003/09/22 by tmartin@tmartin_r400_win

tests the accessability and control of the debug registers

Change 122619 on 2003/09/22 by tmartin@tmartin_r400_win

added the expected output to the test

Change 122598 on 2003/09/22 by tmartin@tmartin_r400_win

test precision of FMT_16_16

Change 122582 on 2003/09/22 by rmanapat@rmanapat_r400_release

Some of the 1D functions were not setting there dimension to 1D thus causing a mismatch.

Change 122573 on 2003/09/22 by ashishs@fl_ashishs_r400_win

Adding test for precision of CUBE instruction. Performing a local CUBE function and calculating the the cube values in the test itself and then passing those output values to the shader and comparing with the output of the shader. This test is different from r400sp_precision_cube_01 in terms of the data set on which it checks on. This test has data set ranging over a larger area of the valid data range.

Change 122535 on 2003/09/22 by mangeshn@fl_mangeshn

added NAN/INF test for the DST instruction. Added source files and updated the test tracker and the SP test list

Change 122519 on 2003/09/22 by mangeshn@fl_mangeshn

added precision test for the DST instruction. Adding source code files, updating the SP test_list and the test tracker

Change 122494 on 2003/09/22 by llefebvr@llefebvr_r400_emu_montreal

now waiting for GFX idle between each pass.

Change 122488 on 2003/09/22 by mangeshn@fl_mangeshn

update

Change 122485 on 2003/09/22 by mangeshn@fl_mangeshn

updated SP test list

Change 122479 on 2003/09/22 by jhoule@jhoule_r400_win_lt

Small fix concerning number of arguments passed

Change 122469 on 2003/09/22 by mangeshn@fl_mangeshn

updating some old tests

Change 122456 on 2003/09/22 by chammer@chammer_xenos_linux_orl

Added RB_TILECONTROL register to list of SC registers which are randomized.

Change 122455 on 2003/09/22 by ashishs@fl_ashishs_r400_win

Sorting the test_list and removing 3 duplicate tests which was causing the total tests to show up as 52 and test run as 49

Change 122453 on 2003/09/22 by mangeshn@fl_mangeshn

pending test checkins and updates to test tracker

Change 122449 on 2003/09/22 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 122442 on 2003/09/22 by tmartin@tmartin_r400_win

added performance counter tests and some format conversion tests

Change 122425 on 2003/09/21 by smoss@smoss_xenos_linux_orl

update

Change 122417 on 2003/09/20 by gregm@fl_gregm

submit

Change 122416 on 2003/09/20 by gregm@fl_gregm

add_prev precision test

Change 122415 on 2003/09/20 by gregm@fl_gregm

submit

Change 122414 on 2003/09/20 by gregm@fl_gregm

add_prev infNan

Change 122412 on 2003/09/20 by gregm@fl_gregm

update

Change 122411 on 2003/09/20 by gregm@fl_gregm

vector fract infNan

Change 122390 on 2003/09/19 by gregm@fl_gregm

scalar fract nan/inf

Change 122387 on 2003/09/19 by gregm@fl_gregm

update

Change 122385 on 2003/09/19 by gregm@fl_gregm

update

Change 122384 on 2003/09/19 by gregm@fl_gregm

scalar fract inf/nan

Change 122382 on 2003/09/19 by gregm@fl_gregm

Fixed problem that effected random number generation.

Change 122349 on 2003/09/19 by ashishs@fl_ashishs_r400_win

adding tests to test_list

Change 122346 on 2003/09/19 by ashishs@fl_ashishs_r400_win

Adding test for precison of Cube. Has 512 vectors over whioch the data is checked, Each vector with random x,y,z,w ranging between -100000000 to +100000000. Built a function to calculate cube value before hand and pass that value to the shaders for comparison with the shader generated value

Change 122332 on 2003/09/19 by mkelly@fl_mkelly_r400_win_laptop

Save work in progress...

Change 122317 on 2003/09/19 by domachi@domachi_xenos

Ensure msaa num samples is not 0 when msaa is enabled. Fixes assert in emulator seen with this test.

Change 122305 on 2003/09/19 by tmartin@tmartin_r400_win

added printing of select values and cleaned up files

Change 122296 on 2003/09/19 by mangeshn@fl_mangeshn

added INF/NAN data test for the SUB_PREV instruction

Change 122290 on 2003/09/19 by mangeshn@fl_mangeshn

added precision test for the SUB_PREV instruction

Change 122278 on 2003/09/19 by ashishs@fl_ashishs_r400_win

Also adding the ability to dump data in image, but not actually dumping the data to keep the iamge same as earlier

Change 122275 on 2003/09/19 by georgev@devel_georgev_r400_lin2_marlboro_tott

Added extra swizzle case for combination values.

Change 122264 on 2003/09/19 by ashishs@fl_ashishs_r400_win

just getting the test cases to 512 to support the template and adding dummy data at the end

Change 122263 on 2003/09/19 by mangeshn@fl_mangeshn

added test for INF/NAN data for the SUB instruction

Change 122258 on 2003/09/19 by domachi@diotargetxp

- Changes to do some TP random testing. Only Fetch Constants have randomized values. Still need to add support to randomize Fetch Instruction values.

- Reverted a change where base map mip filtering was tested thereby increasing test times. This change should reduce test times.

Change 122255 on 2003/09/19 by ashishs@fl_ashishs_r400_win

Changed the precison cosnt to 2**21 so the image will change a little bit but has been validated to be okay.

Also changed the test to the new template to cause the instruction to work on all the shader pipes as well as generalising a little bit so that we could just run the case number needed

Change 122240 on 2003/09/19 by mkelly@fl_mkelly_r400_win_laptop

Save work in progress, data dependent vertex fetching...

Change 122187 on 2003/09/19 by smoss@smoss_xenos_linux_orl

update for tb_vc

Change 122155 on 2003/09/19 by jayw@jayw_r400_linux_marlboro2

Fix TB

Change 122107 on 2003/09/18 by eberger@eberger_r400_linux_marlboro

Added code to set the WINDOW_SCISSOR correctly. Added comments.

Change 122070 on 2003/09/18 by tmartin@tmartin_r400_win

FMT_11_11_10

Change 122069 on 2003/09/18 by tmartin@tmartin_r400_win

FMT_10_11_11

Change 122056 on 2003/09/18 by georgev@devel_georgev_r400_lin2_marlboro_tott

Border tests with and without W forced to max. Formats added to tp_simple_format_test.

Change 122042 on 2003/09/18 by mkelly@fl_mkelly_r400_win_laptop

Update for new SPI.

Change 122040 on 2003/09/18 by mkelly@fl_mkelly_r400_win_laptop

Updates to work with new SPI.

Change 122038 on 2003/09/18 by tmartin@tmartin_r400_win

performance counter tests

Change 122028 on 2003/09/18 by mkelly@fl_mkelly_r400_win_laptop

Update to work in the new SPI.

Change 122021 on 2003/09/18 by domachi@domachi_xenos

Checkpoint changes to random tests.

Change 122018 on 2003/09/18 by mkelly@fl_mkelly_r400_win_laptop

Change for new SPI.

Change 122015 on 2003/09/18 by tmartin@tmartin_r400_win

tests precision of fmt_2_10_10_10

Change 122013 on 2003/09/18 by tmartin@tmartin_r400_win

tests precision of fmt_8_8_8_8

Change 122001 on 2003/09/18 by mkelly@fl_mkelly_r400_win_laptop

Change shader to work with new SPI. Add catch in test for debugging help.

Change 121986 on 2003/09/18 by mkelly@fl_mkelly_r400_win_laptop

Update test to work with new SPI implementation on prim type detection in the pixel shader.

Change 121985 on 2003/09/18 by mkelly@fl_mkelly_r400_win_laptop

Increase NUM_CASES = 64 from 9

Change 121976 on 2003/09/18 by tmartin@tmartin_r400_win

set clamp_disable to 1

Change 121975 on 2003/09/18 by mkelly@fl_mkelly_r400_win_laptop

Data dependent predicated fetch of secondary data buffer.

Change 121963 on 2003/09/18 by smoss@smoss_xenos_linux_orl

updated for new tree

Change 121914 on 2003/09/17 by kevino@kevino_r400_linux_marlboro

Fix tp_mutlitexture_02 to have all formats (have a define in there to only do fmt06 to make build faster when developing tests.)
Added basemap, tcdenorm, and tp_multitexture_02 fmt02 tests to regressions.

Change 121910 on 2003/09/17 by ashishs@fl_ashishs_r400_win

Currently just doing 44 bad test cases

Change 121889 on 2003/09/17 by smoss@smoss_xenos_linux_orl

added better failure reporting, new output path

Change 121885 on 2003/09/17 by ashishs@fl_ashishs_r400_win

changing the data since had incorrect values

Change 121882 on 2003/09/17 by gregm@fl_gregm

Special infNan test for add instruction and vertex cache.

Change 121881 on 2003/09/17 by gregm@fl_gregm

update

Change 121879 on 2003/09/17 by kevino@kevino_r400_linux_marlboro

Test to put the texture basemap right at the upper edge of memory (where tc_BaseMapOffset couldn't reach)

Change 121878 on 2003/09/17 by kevino@kevino_r400_linux_marlboro

Updated tp_multitexture_01 and _02 to draw framebuffer correctly. Added tcdenorm tests. Added some new texture maps to data/tex. Updated a few of the 3d testcases

Change 121829 on 2003/09/17 by georgev@devel_georgev_r400_lin2_marlboro_tott

New file because last one had bad format.

Change 121760 on 2003/09/17 by mkelly@fl_mkelly_r400_win_laptop

Move STATE read data out of framebuffer.

Change 121729 on 2003/09/17 by mkelly@fl_mkelly_r400_win_laptop

Update for SPI

Change 121723 on 2003/09/17 by mkelly@fl_mkelly_r400_win_laptop

Update for SPI

Change 121709 on 2003/09/17 by tmartin@tmartin_r400_win

This test fills the Index fifo to test its handling of overflows

Change 121706 on 2003/09/17 by mkelly@fl_mkelly_r400_win_laptop

Change for SPI

Change 121705 on 2003/09/17 by mkelly@fl_mkelly_r400_win_laptop

Changes for new SPI...

Change 121704 on 2003/09/17 by mkelly@fl_mkelly_r400_win_laptop

Add RECIP for faceness/XY detection in SPI.

Change 121676 on 2003/09/16 by csampayo@fl_csampayo_r400

Adding pixel shader export test
Updated test_list and test tracker accordingly

Change 121669 on 2003/09/16 by ashishs@fl_ashishs_r400_win2

initial checkin for cube test, simple data set covering all of the statements inside the cube instruction

Change 121667 on 2003/09/16 by tmartin@tmartin_r400_win

This test fills the L2 request fifo to test its handling of overflows

Change 121637 on 2003/09/16 by domachi@diotargetxp

Add random shader generation to sc_rand test

Change 121634 on 2003/09/16 by mkelly@fl_mkelly_r400_win_laptop

SERIALIZE properly...

Change 121632 on 2003/09/16 by llefebvr@llefebvre_laptop_r400_emu

Made the XYs be sent as floating point numbers instead of fix point so you can use them directly in the shader. Also modified regress_e tests that this change broke. Also added register fields for SIMD memory control.

Change 121628 on 2003/09/16 by mkelly@fl_mkelly_r400_win_laptop

Properly SERIALIZE...

Change 121626 on 2003/09/16 by mkelly@fl_mkelly_r400_win_laptop

SERIALIZE properly.

Change 121620 on 2003/09/16 by mkelly@fl_mkelly_r400_win_laptop

Add SERIALIZE correctly.

Change 121618 on 2003/09/16 by mkelly@fl_mkelly_r400_win_laptop

Need SERIALIZE after all fetch routines before first ALU dependent on fetch.

Change 121615 on 2003/09/16 by mkelly@fl_mkelly_r400_win_laptop

Remove gpr allocation from shaders...

Change 121606 on 2003/09/16 by georgev@devel_georgev_r400_lin2_marlboro_tott

Added extra formats. Began conversion for max W registers.

Change 121572 on 2003/09/16 by mkelly@fl_mkelly_r400_win_laptop

Window and screen scissor set to 8192,32 render two points
1 point at X = 8190, 1 point at X = 8191.

Change 121551 on 2003/09/16 by mkelly@fl_mkelly_r400_win_laptop

2 vert POINT_LIST test case for scissor BR X = 8192, BR Y = 32

Change 121550 on 2003/09/16 by jhoule@jhoule_r400_win_lt

Meaningless space to avoid make error

Change 121519 on 2003/09/16 by tmartin@tmartin_r400_win

added more float4's to the vfetches

Change 121499 on 2003/09/16 by jhoule@jhoule_r400_win_lt

Reactivated aniso

Change 121496 on 2003/09/16 by jhoule@jhoule_r400_win_lt

New vertex shader program file

Change 121495 on 2003/09/16 by jhoule@jhoule_r400_win_lt

New textures (for aniso tests)

Change 121494 on 2003/09/16 by jhoule@jhoule_r400_win_lt

Putting simple aniso test files

Change 121418 on 2003/09/15 by tmartin@tmartin_r400_win

added r400vc_simple_register_indirect and r400vc_fifo_l1_req_01

Change 121417 on 2003/09/15 by tmartin@tmartin_r400_win

fixed some typos

Change 121412 on 2003/09/15 by tmartin@tmartin_r400_win

This test fills the L1 request fifo to test its handling of overflows

Change 121405 on 2003/09/15 by eberger@eberger_r400_linux_marlboro

Turned off the depth buffer whenever multisampling is enabled.

Change 121392 on 2003/09/15 by tmartin@tmartin_r400_win

Register Writes/Reads

Change 121384 on 2003/09/15 by mkelly@fl_mkelly_r400_win_laptop

Test carrying the (i) loop index down through nested subroutines, data dependecy predication.

Change 121379 on 2003/09/15 by vromaker@vromaker_r400_linux_marlboro

added SERIALIZE commands within the subroutines

Change 121368 on 2003/09/15 by eberger@eberger_r400_linux_marlboro

Added tile_surface.Dump() and cleaned up some other problems.

Change 121365 on 2003/09/15 by mangeshn@fl_mangeshn

test updates

Change 121341 on 2003/09/15 by ashishs@fl_ashishs_r400_win

updated description of some tests. Also still some TBD's left but need to revise the test after running them and then remove these descriptions.

Change 121329 on 2003/09/15 by mkelly@fl_mkelly_r400_win_laptop

Data dependent predicate pop,inv,restore,clr nested looping and subroutines.

Change 121316 on 2003/09/15 by mkelly@fl_mkelly_r400_win_laptop

Update to correct shader...

Change 121305 on 2003/09/15 by mangeshn@fl_mangeshn

added precision test for the DOT2ADD instruction

Change 121304 on 2003/09/15 by mkelly@fl_mkelly_r400_win_laptop

del unnecessary file

Change 121303 on 2003/09/15 by mkelly@fl_mkelly_r400_win_laptop

Update comments in _11
Further looping index testing with nested predication and subroutines.

Change 121289 on 2003/09/15 by jhoule@jhoule_r400_win_lt

Adding Set* opcodes.

tp_lod_deriv:
- Added inputs (is_set_lod_command, is_set_grad_command, pix_mask, p3xyz)
- Updated the parser and testbench program accordingly

TexturePipe:
- Added overrides for SetTexLOD and SetRegGradients in ComputeLODs_V2
- Put placeholder for other opcodes

Change 121275 on 2003/09/15 by mkelly@fl_mkelly_r400_win_laptop

Nested looping with predicated execution, boolean calling subroutines.

Change 121274 on 2003/09/15 by ashishs@fl_ashishs_r400_win

correcting the FAILING tests. These tests were sending uninitialised data thru the alpha channel due to which initially it was getting clamped to 1 and now it gets clamped to 0. So initialising the alpha of verts to 1.0

Change 121235 on 2003/09/14 by gregm@fl_gregm

update

Change 121234 on 2003/09/14 by gregm@fl_gregm

precision scalar add

Change 121229 on 2003/09/14 by gregm@fl_gregm

update

Change 121228 on 2003/09/14 by gregm@fl_gregm

scalar add

Change 121225 on 2003/09/14 by gregm@fl_gregm

update

Change 121224 on 2003/09/14 by gregm@fl_gregm

r400sp_infNan_add_01 (vector add)

Change 121095 on 2003/09/12 by mangeshn@fl_mangeshn

added precision test for the SUB instruction

Change 121089 on 2003/09/12 by mangeshn@fl_mangeshn

added precision test for the DOT3 instruction

Change 121085 on 2003/09/12 by tmartin@tmartin_r400_win

Tests vertex reuse in the VC. A strip is drawn with a triangle list using indices mulitple times in the index list. Vertex reuse in the VGT is disabled to ensure the only reuse is in the VC.

Change 121079 on 2003/09/12 by mangeshn@fl_mangeshn

completed version of teh dot4 precision test. points that are red have a bit difference NOT in the LSB. Blue points have a bit difference in teh LSB. Green points are exact matches

Change 121049 on 2003/09/12 by lseiler@lseiler_r400_win_marlboro

Fix windows compile bug due to multiply defined for-loop variable

Change 121045 on 2003/09/12 by eberger@eberger_r400_linux_marlboro

Initial version of a test for a frame buffer with maximum pitch.

Change 121044 on 2003/09/12 by kevino@kevino_r400_linux_marlboro

Test with 1 case that pushes the base texture map to 8K before 512MB. Couldn't get it to 512MB - 4K. Not sure why 32x32 32 bit texture takes > 4k even if it is linear.

Change 121025 on 2003/09/12 by mkelly@fl_mkelly_r400_win_laptop

Save work in progress...

Change 121011 on 2003/09/12 by lseiler@lseiler_r400_win_marlboro2

New rb-specific tests

Change 121008 on 2003/09/12 by lseiler@lseiler_r400_win_marlboro2

New rb-specific tests

Change 121004 on 2003/09/12 by kevino@kevino_r400_linux_marlboro

Varies offset of texture Basemap from 0 to just under 512MB.

Change 120977 on 2003/09/12 by lseiler@lseiler_r400_win_marlboro1

Update local gold images (regress_rb)

Change 120976 on 2003/09/12 by lseiler@lseiler_r400_win_marlboro1

Update local gold image list (regress_rb)

Change 120936 on 2003/09/12 by smoss@smoss_crayola_linux_orl

new output directories

Change 120901 on 2003/09/12 by mkelly@fl_mkelly_r400_win_laptop

Single case, 4th nested loop clamping issue.

Change 120817 on 2003/09/11 by tmartin@tmartin_r400_win

cleaned up the tests

Change 120813 on 2003/09/11 by tmartin@tmartin_r400_win

added r400vc_size_01

Change 120810 on 2003/09/11 by tmartin@tmartin_r400_win

Checks the cache size and makes sure the cache ways work as expected

Change 120784 on 2003/09/11 by csampayo@fl_csampayo2_r400

Clen up and finalize for pixel exports

Change 120776 on 2003/09/11 by omesh@omesh_r400_linux_marlboro_tott

Fixed a bug with setting the full device start address for the resolved
surface. Earlier the resolved image appeared black, as reported by
Larry, but now it does produce a non-black pixel.

Change 120768 on 2003/09/11 by csampayo@fl_csampayo_r400

Initial checkin

Change 120718 on 2003/09/11 by mkelly@fl_mkelly_r400_win_laptop

Changed write from VGT_EVENT_INITIATOR for VS_DONE_TS to an EVENT_WRITE packet3 type.

Change 120703 on 2003/09/11 by mangeshn@fl_mangeshn

changed test to showcase a set of 44 failing points

Change 120667 on 2003/09/11 by mkelly@fl_mkelly_r400_win_laptop

SET_STATE, creates state buffers, writes them to framebuffer memory,
then initiates a SET_STATE packet with pointers to the state buffers.

Change 120594 on 2003/09/10 by kevino@kevino_r400_linux_marlboro

Don't write temp sp files to subdirectory- seemed to somehow write to subdir AND base dir.

Change 120591 on 2003/09/10 by georgev@devel_georgev_r400_lin2_marlboro_tott

Fixed perspective for higher aniso ratios.

Change 120575 on 2003/09/10 by ashishs@fl_ashishs_r400_win

adding 2 more tests

Change 120566 on 2003/09/10 by ashishs@fl_ashishs_r400_win

test using 64 shaders with each shader uniquely exportin a chosen set of parameters and then using sampling pattern as cetroids on the exported parameters and rest as centers

Change 120554 on 2003/09/10 by ashishs@fl_ashishs_r400_win

removing gfx_idle_no_flush

Change 120548 on 2003/09/10 by ashishs@fl_ashishs_r400_win

test using 64 shaders with each shader uniquely exportin a chosen set of parameters and then using sampling pattern as centers on the exported parameters and rest as centroids

Change 120527 on 2003/09/10 by tmartin@tmartin_r400_win

added tfetches to shaders in order to test gpr_phase

Change 120526 on 2003/09/10 by mangeshn@fl_mangeshn

added precision test for the DOT4 instruction

Change 120525 on 2003/09/10 by kevino@kevino_r400_linux_marlboro

Add setalphatograyscale to tp_multitexture_02 for cases where textures are read from file (as opposed to build, where alpha does get valid data)

Change 120524 on 2003/09/10 by kevino@kevino_r400_linux_marlboro

For tests that create their own pixel shader files, make a unique name for each testcase so they won't overwrite eachotehr when multiple testcases are running at once.

Change 120499 on 2003/09/10 by ashishs@fl_ashishs_r400_win

added 2 new tests

Change 120495 on 2003/09/10 by mkelly@fl_mkelly_r400_win_laptop

Vary packet size, conditional predicated execution max subs and loops.

Change 120470 on 2003/09/10 by omesh@omesh_r400_linux_marlboro_tott

Fixed resolve tests so that the single render state being used to dump
both multisampled as well as resolved surfaces does not cause incorrect
header information (and emulator/primlib assertions) from happening. The
tests now run to completion.

Change 120435 on 2003/09/10 by mangeshn@fl_mangeshn

added precision test for checking NAN/INF cases for the EXP_IEEE instruction and updated the tracker

Change 120422 on 2003/09/10 by jhoule@jhoule_r400_win_lt

Added support for local testsuite with testcases.

Updated the golden_tc.lst file to support the following test:
r400rb_stencil_functions-STENCIL_STENCILFUNC_GREATER_1_S.regress_e

Adding new test should be a matter of adding lines to CHIP_TESTS_TC and keeping the golden_tc.lst up-to-date (note: if the test is not present in the golden_tc.lst file, it won't run).

Change 120409 on 2003/09/10 by mkelly@fl_mkelly_r400_win_laptop

Pixel shader conditional predicate execute contained in maximum nested subroutines and loops.

Change 120344 on 2003/09/09 by tmartin@tmartin_r400_win

test coherency regs, RT and HOST

Change 120343 on 2003/09/09 by tmartin@tmartin_r400_win

test coherency regs, RT and HOST

Change 120338 on 2003/09/09 by ashishs@fl_ashishs_r400_win

changing the orientation of the primitive....number of samples to 2 and randomly switching of sampling_patter in interpolator_cntl

Change 120326 on 2003/09/09 by tmartin@tmartin_r400_win

added tests for VC_CNTL and coherency

Change 120320 on 2003/09/09 by mangeshn@fl_mangeshn

added new tests to SP test_list

Change 120312 on 2003/09/09 by tmartin@tmartin_r400_win

changed test names

Change 120311 on 2003/09/09 by tmartin@tmartin_r400_win

changed test names

Change 120307 on 2003/09/09 by tmartin@tmartin_r400_win

changed the test name

Change 120305 on 2003/09/09 by ashishs@fl_ashishs_r400_win

permuting INTERPOLATOR_CNTL.SAMPLING_PATTERN with SC output as centers and centroids

Change 120299 on 2003/09/09 by mkelly@fl_mkelly_r400_win_laptop

Conditional execute predication in maximum nested subroutines using vertex stream data.
Exercises (!P) where r400sq_data_dep_pred_05 did (P).

Change 120287 on 2003/09/09 by mangeshn@fl_mangeshn

added precision test to handle NAN/INF cases for the LOG_CLAMPED instruction. Also updated the Test Tracker doc

Change 120278 on 2003/09/09 by mkelly@fl_mkelly_r400_win_laptop

Maximum nested loops with data dependent predication execution contained
within maximum nested subroutines.

Change 120276 on 2003/09/09 by cbrennan@cbrennan_r400_emu

Remove restriction of disallowing degamma of new DXT formats in random_level4.
Wont actually validate unless #ifdef C1 is defined in tex lib.
Added degamma test cases for DXN, DXT3a, DXT5a, and CTX1. Dont expect to pass use unless kinky degamma is enabled.

Change 120255 on 2003/09/09 by lseiler@lseiler_r400_win_marlboro2

Chroma key gold files

Change 120254 on 2003/09/09 by jhoule@jhoule_r400_linux_marlboro

Added per-channel precision loss in the tp_ch_blend subblock.
Precision out of the bilinear lerps are now 20, 20, 16, and 12.
Output becomes 30, 30, 26, 22 (10 more than the above); better in hardware too.
Added setPreWsPrecision() method to tp_ch_blend to control precision.

Modified tp_tt to account for precision loss (calls get30bOutput() on the tp_ch_blend).

Created a new parser which has 3 tp_ch_blend, each one with a different precision.

Updated the dedicated testbench to use that new parser.

Change 120240 on 2003/09/09 by mangeshn@fl_mangeshn

added precision test for INF/NAN checking for the LOG_IEEE instruction

Change 120222 on 2003/09/09 by jhoule@jhoule_r400_win_lt

Added MIN/MAG_ANISO_WALK support in the tp_lod_aniso.
Updated testbench ins+outs, as well as tracker outputs.

Change 120221 on 2003/09/09 by lseiler@lseiler_r400_win_marlboro

Depth test GOLD files

Change 120197 on 2003/09/09 by mkelly@fl_mkelly_r400_win_laptop

Pixel Shader conditional predicate execute in maximum nested subroutines, dependent on parameters originated from the vertex stream.

Change 120181 on 2003/09/09 by mangeshn@fl_mangeshn

added precision test to handle NAN/INF cases for teh RECIP_SQRT_FF instruction

Change 120148 on 2003/09/09 by ashishs@fl_ashishs_r400_win

corrected a test name

Change 120143 on 2003/09/09 by lseiler@lseiler_r400_win_marlboro1

Zpass test

Change 120142 on 2003/09/09 by lseiler@lseiler_r400_win_marlboro1

Stencil Zfail test

Change 120082 on 2003/09/08 by tmartin@tmartin_r400_win

added some interface tests

Change 120079 on 2003/09/08 by tmartin@tmartin_r400_win

Tests CC_FORCE_MISS

Change 120078 on 2003/09/08 by tmartin@tmartin_r400_win

Tests L2_INVALIDATE

Change 120077 on 2003/09/08 by tmartin@tmartin_r400_win

Tests the thread id's by performing vfetches in both shader program types

Change 120070 on 2003/09/08 by mangeshn@fl_mangeshn

added precision test to handle INF/NAN cases for the RECIP_SQRT_CLAMPED instruction

Change 120059 on 2003/09/08 by mangeshn@fl_mangeshn

added precision test to handle INF/NAN cases for teh RECIP_SQRT_IEEE instruction

Change 120056 on 2003/09/08 by mangeshn@fl_mangeshn

added precision test to handle INF/NAN for teh SQRT_IEEE instruction

Change 120025 on 2003/09/08 by lseiler@lseiler_r400_win_marlboro1

Stencil test gold images

Change 119989 on 2003/09/08 by mkelly@fl_mkelly_r400_win_laptop

Conditional execute predication in maximum nested
subroutines using vertex stream data. Exercise (!P) as
compare to _01 that exercises (P).

Change 119977 on 2003/09/08 by kevino@kevino_r400_linux_marlboro

Same test as tp_mutlitexture_01, but now all formats are supported and testcase names use format number instead of name. This will eventually replace tp_mutlitexture_01. Not in this version, interlaced formats do not work.

Change 119976 on 2003/09/08 by mkelly@fl_mkelly_r400_win_laptop

Conditional execute predication in maximum nested subroutines using vertex stream data.

Change 119958 on 2003/09/08 by mangeshn@fl_mangeshn

added new tests to test_lib

Change 119956 on 2003/09/08 by mangeshn@fl_mangeshn

added precision test for the PRED_SETE instruction

Change 119925 on 2003/09/08 by kevino@kevino_r400_linux_marlboro

Added setAlphaToGrayscale() to tp_simple_01 and _02

Change 119916 on 2003/09/08 by jayw@jayw_r400_linux_marlboro2

Weekly re-integration. sc has changed, need new one for GC TB.

Change 119888 on 2003/09/07 by mangeshn@fl_mangeshn

added precision tests for CNDE, CNDGE and CNDGT instructions

Change 119866 on 2003/09/06 by mangeshn@fl_mangeshn

added precision test for the KILLNE instruction for VECTOR operations

Change 119865 on 2003/09/06 by mangeshn@fl_mangeshn

added precision test for the KILLNE instruction for SCALAR operations

Change 119864 on 2003/09/06 by mangeshn@fl_mangeshn

added precision test for the KILLGE instruction for VECTOR operations

Change 119863 on 2003/09/06 by mangeshn@fl_mangeshn

added precision test for the KILLGE instruction for SCALAR operations

Change 119862 on 2003/09/06 by mangeshn@fl_mangeshn

added precision test for the KILLGT instruction for the vector operation

Change 119860 on 2003/09/06 by mangeshn@fl_mangeshn

added precision test for the KILLGT instruction for SCALAR operation

Change 119855 on 2003/09/06 by mangeshn@fl_mangeshn

adde precision test for KILLE instruction for the VECTOR operation

Change 119854 on 2003/09/06 by mangeshn@fl_mangeshn

added precision test for the KILLE instruction. SCALAR evaluation only

Change 119828 on 2003/09/06 by mangeshn@fl_mangeshn

added precision test for the SETNE instruction for both the SCALAR and VECTOR forms

Change 119826 on 2003/09/06 by mangeshn@fl_mangeshn

added precision test for the SETGE instruction for both the SCALAR and VECTOR operations.

Change 119825 on 2003/09/06 by mangeshn@fl_mangeshn

added precision test for the SETGT instruction. Handles SCALAR and VECTOR operations.

Change 119808 on 2003/09/05 by mangeshn@fl_mangeshn

modified the SETE test to check for both SCALAR and VECTOR operations (each channel is tested individually)

Change 119782 on 2003/09/05 by tmartin@tmartin_r400_win

added test r400vc_instr_fields_10 for PRED_SELECT and PRED_CONDITION in the vfetch instruction

Change 119755 on 2003/09/05 by mangeshn@fl_mangeshn

added precision test for the SCALAR sete instruction

Change 119723 on 2003/09/05 by mkelly@fl_mkelly_r400_win_laptop

Save work in progress.

Change 119664 on 2003/09/05 by tmartin@tmartin_r400_win

Tests PRED_SELECT and PRED_CONDITION in the vfetch instruction

Change 119660 on 2003/09/05 by mangeshn@fl_mangeshn

added a precision test for inf/nan handling for the recip_clamp instruction

Change 119630 on 2003/09/05 by mangeshn@fl_mangeshn

changed the old test template to the new one with 512 points

Change 119582 on 2003/09/04 by mangeshn@fl_mangeshn

changed test to new framework and colors to previous tests

Change 119551 on 2003/09/04 by mangeshn@fl_mangeshn

changed color scheme of output to match previous tests

Change 119549 on 2003/09/04 by tmartin@tmartin_r400_win

added r400vc_instr_fields_07 and 09

Change 119548 on 2003/09/04 by mangeshn@fl_mangeshn

changed colors in result to match earlier tests

Change 119546 on 2003/09/04 by tmartin@tmartin_r400_win

Tests SRC_GPR_AM and DST_GPR_AM

Change 119539 on 2003/09/04 by vbhatia@vbhatia_r400_linux_marlboro

cube map related update to standalone lod_deriv testbench

Change 119523 on 2003/09/04 by mangeshn@fl_mangeshn

adding mangesh and greg

Change 119521 on 2003/09/04 by mangeshn@fl_mangeshn

updated

Change 119505 on 2003/09/04 by mangeshn@fl_mangeshn

Added test with INF and NAN values along with valid range data tests for recip_ff instruction

Change 119477 on 2003/09/04 by ashishs@fl_ashishs_r400_win2

the following tests were failing since they had JSS register setting

Change 119464 on 2003/09/04 by mkelly@fl_mkelly_r400_win_laptop

Save.

Change 119442 on 2003/09/04 by ashishs@fl_ashishs_r400_win

Editing the test so that we can create a template test which can be used for other instructions

Change 119424 on 2003/09/04 by mkelly@fl_mkelly_r400_win_laptop

Different tests now than before because the random seed has changed
due to removal of jss. It is all random based.

Change 119405 on 2003/09/04 by tmartin@tmartin_r400_win

Tests SIGNED_RF_MODE_ALL

Change 119370 on 2003/09/04 by cbrennan@cbrennan_r400_emu

Changed tests to target cube mapping range from 1.0 to 2.0

Change 119314 on 2003/09/03 by smoss@smoss_crayola_linux_orl

typos in test list, removed jss from rand test

Change 119306 on 2003/09/03 by tmartin@tmartin_r400_win

added some instruction fields tests

Change 119305 on 2003/09/03 by jhoule@jhoule_r400_win_lt

Changed variable name to better behave with Frank Hsien's script

Change 119300 on 2003/09/03 by tmartin@tmartin_r400_win

ests the range of EXP_ADJUST_ALL (-32 to 31)

Change 119299 on 2003/09/03 by tmartin@tmartin_r400_win

Tests the upper 16 values for CONST_INDEX and all CONST_INDEX_SEL values. The lower 16 values are tested in instr_fields_01 - 04. This test builds on r400vc_instr_fields_01

Change 119292 on 2003/09/03 by ashishs@fl_ashishs_r400_win2

updated

Change 119282 on 2003/09/03 by ashishs@fl_ashishs_r400_win2

updated the test to finalise the template which could be used to test a varied data range for the instruction

Change 119228 on 2003/09/03 by jhoule@jhoule_r400_win_lt

Added regress_rb target to be used by Larry to run a more exhaustive emulator regression before submitting new code (some tests are currently failing).

Tests must be listed in the LOCAL_CHIP_TESTS variable (which doesn't seem to support testcases for now).
Uses regress_e_local target, which is less rb-specific.

Change 119200 on 2003/09/03 by tmartin@tmartin_r400_win

Tests SRC_SEL in the vertex fetch instruction

Change 119125 on 2003/09/02 by ashishs@fl_ashishs_r400_win

changing the test so that it renders 128 verts in a packet at a time so that it exercises all the shader pipes

Change 119119 on 2003/09/02 by tmartin@tmartin_r400_win

added tests for constant and instruction fields

Change 119118 on 2003/09/02 by tmartin@tmartin_r400_win

Tests GPRs as both source and destination

Change 119103 on 2003/09/02 by ashishs@fl_ashishs_r400_win

initial checkin for the template test to check each instruction for inf,Nan,denorm,special numbers along with valid data range

Change 119044 on 2003/09/02 by tmartin@tmartin_r400_win

Tests 4 bit patterns in the base_address field. 0x55555554, 0xAAAAAAA8, 0x00007FFC, 0xFFFF0000

Change 118842 on 2003/08/29 by jhoule@jhoule_r400_linux_marlboro

Simple test allowing to easily test all formats.
Simpler than the currently existing ones.

Change 118837 on 2003/08/29 by ashishs@fl_ashishs_r400_win

adding r400sq_mova_01 after it has started PASSING now

Change 118834 on 2003/08/29 by ashishs@fl_ashishs_r400_win

adding golds to the SQ directory using make <test_name>.golden

Change 118831 on 2003/08/29 by cbrennan@cbrennan_r400_emu

compile fix

Change 118830 on 2003/08/29 by cbrennan@cbrennan_r400_win_marlboro

Added seed of a test to check new cube mapping lod changes.

Change 118816 on 2003/08/29 by tmartin@tmartin_r400_win

fixed a typo

Change 118810 on 2003/08/29 by csampayo@fl_csampayo2_r400

Cleanup shaders

Change 118790 on 2003/08/29 by ashishs@fl_ashishs_r400_p4D

updated the test_list to disable r400sq_vs_memory_wrap_01 test since hangs in emulator

Change 118779 on 2003/08/29 by ashishs@fl_ashishs_r400_win

forgot to enable the option -u in the script

Change 118754 on 2003/08/29 by llefebvr@llefebvr_r400_emu_montreal

Fixing shader

Change 118752 on 2003/08/29 by csampayo@fl_csampayo2_r400

Adjus test to remove texture usage in pixel shader

Change 118740 on 2003/08/29 by tmartin@tmartin_r400_win

added data_format tests

Change 118737 on 2003/08/29 by tmartin@tmartin_r400_win

test all vertex data formats and stress all pipes

Change 118716 on 2003/08/29 by georgev@devel_georgev_r400_lin2_marlboro_tott

Severely stretched all images.

Change 118698 on 2003/08/29 by jhoule@jhoule_r400_win_lt

Added support for setAlphaToGrayscale()

Change 118582 on 2003/08/28 by omesh@omesh_r400_linux_marlboro_tott

Fixed testcases that expand previously rendered to surfaces that use
multisampling. During expands, surface origins must be adjusted to the
top left corner of the top left pixel, rather than keeping it at the
center of the top left pixel. (The former results in SC generating out
of bound addresses towards RC/RB). Verified that the specific testcase
that Jay pointed out does not assert any longer.

Change 118565 on 2003/08/28 by georgev@devel_georgev_r400_lin2_marlboro_tott

Put in more perspective so that aniso would work better.

Change 118550 on 2003/08/28 by ashishs@fl_ashishs_r400_win

Disabling 2 of the jss tests.

r400sc_point_jss_3X4_01
r400sc_line_jss_3X4_01

Change 118547 on 2003/08/28 by mkelly@fl_mkelly_r400_win_laptop

Omit resolve tests from regress for now until new RB ready.

Change 118521 on 2003/08/28 by mkelly@fl_mkelly_r400_win_laptop

Stress the parameter cache with clipped verts and many small triangles...

Change 118466 on 2003/08/27 by csampayo@fl_csampayo2_r400

1 Modified constant 0 in both cpp files
2 Further cleaned up of _01 cpp and vtx

Change 118445 on 2003/08/27 by tmartin@tmartin_r400_win

finished the remaining formats

Change 118436 on 2003/08/27 by ashishs@fl_ashishs_r400_win

adding VS and PS base to the test

Change 118420 on 2003/08/27 by ashishs@fl_ashishs_r400_win

changing the pix and vtx shaders to ver 2.0 and adding alloc for paramteres as well

Change 118355 on 2003/08/27 by csampayo@fl_csampayo2_r400

Revert back to latest version, add alloc inst to vtx shader

Change 118349 on 2003/08/27 by ashishs@fl_ashishs_r400_win

setting IM_LOAD inside the tests

Change 118344 on 2003/08/27 by ashishs@fl_ashishs_r400_win

changed pixel shader to revert back to the original shader viz. Even if vtx shader exporting 3 parameters, pixel shader only fetching the parameter needed

Change 118273 on 2003/08/26 by csampayo@fl_csampayo_r400

Temporarily go back to rev 2 to catch problem sync

Change 118269 on 2003/08/26 by tmartin@tmartin_r400_win

the wrong shader was being used

Change 118262 on 2003/08/26 by csampayo@fl_csampayo_r400

Remove commented blocks and update description

Change 118255 on 2003/08/26 by ashishs@fl_ashishs_r400_win

updated the test to include IM_LOAD function since causing problem in hardware

Change 118251 on 2003/08/26 by georgev@devel_georgev_r400_lin2_marlboro_tott

Removed illegal gamma cases and added tp_border_02 for forcing w to max.

Change 118224 on 2003/08/26 by omesh@omesh_r400_linux_marlboro_tott

Also set color0 samples, fragments and depth samples and fragments
correctly, before an xpand operation. This should REALLY fix the test
bug. (Not yet verified on simulation though)

Change 118222 on 2003/08/26 by mkelly@fl_mkelly_r400_win_laptop

TRIANGLE_LIST stress tests of exercising parameter cache indices and wrapping with params 1 - 16.

Change 118221 on 2003/08/26 by omesh@omesh_r400_linux_marlboro_tott

Fixed a bug with the testcases that use the xpand function (Which was
not enabling msaa before sending an xpand operation). Haven't verified
this fix on simulation....

Change 118220 on 2003/08/26 by ashishs@fl_ashishs_r400_win

correcting a optimization in the test. No problem with the test just the tests were doing one extra rendering of 128 vertcies which has been removed now.

Change 118214 on 2003/08/26 by ashishs@fl_ashishs_r400_win

forgot to remove a print statement...

Change 118212 on 2003/08/26 by ashishs@fl_ashishs_r400_win

Forgot to initialise the values of the parameters which was causing an error in the script...

Change 118175 on 2003/08/26 by mkelly@fl_mkelly_r400_win_laptop

Extensive line_list testing of the parameter cache, 1-16 parameters.
Update tracker accordingly.

Change 118155 on 2003/08/26 by ashishs@fl_ashishs_r400_win

Perl script Updates:

1. Added option -g
With this option we can check the dumps against the gold directory in each test directory

2. Added option -u
With this option we can update the golds. Currently all the gold files will be updated whichever PASS and for NEW test the gold will be added. Also if the test FAILS then the corresonding gold isn't updated
------Need to know if unchanged files should be reverted back to minimize the depot checkins ?

Change 118153 on 2003/08/26 by mkelly@fl_mkelly_r400_win_laptop

Update expected results comments in test for new constant index behaviour.

Change 118143 on 2003/08/26 by mkelly@fl_mkelly_r400_win_laptop

Remove resolve tests until new RB ready.
Fix line tests due to RB assert when msaa = true and samples = 0.

Change 118120 on 2003/08/25 by ashishs@fl_ashishs_r400_win

removing a duplicate test name

Change 118100 on 2003/08/25 by csampayo@fl_csampayo2_r400

Correct number of loaded constants.

Change 118082 on 2003/08/25 by rmanapat@rmanapat_r400_sun_marlboro

Tests changed so that fmts58, 59, and 60 do not use gamma

Change 118067 on 2003/08/25 by ashishs@fl_ashishs_r400_win

adding marcos to nightly regressions

Change 118051 on 2003/08/25 by jhoule@jhoule_r400_linux_marlboro

Changed VFetch format from 32x2+32x1 to 32+32+32 type.

Change 118050 on 2003/08/25 by mkelly@fl_mkelly_r400_win_laptop

Exercise parameter cache indices with LINE_LIST and TRIANGLE_LIST, 16 parameters, many verts.

Change 118009 on 2003/08/25 by mkelly@fl_mkelly_r400_win_laptop

Remainder of POINT_LIST paramater cache testing of the memory addressing and indice
generation.

Change 118000 on 2003/08/25 by ashishs@fl_ashishs_r400_win

checking in all the auto generated shader files by VFD with modification regarding the exports viz exports should be sequential starting at 0 and if a parameter is exported on higher export register then need to export all the lower registers too or else doesnt work with hardware

Change 117969 on 2003/08/25 by mkelly@fl_mkelly_r400_win_laptop

POINT_LIST, 16 params, exercise the full range of PC addresses with
address wrapping.

Change 117870 on 2003/08/22 by csampayo@fl_csampayo_r400

Partial update

Change 117865 on 2003/08/22 by tmartin@tmartin_r400_win

added r400vc_addr_clamping_02

Change 117855 on 2003/08/22 by ashishs@fl_ashishs_r400_win

corrected the tests since i forgot to use the 4th shader while rendering

Change 117851 on 2003/08/22 by ashishs@fl_ashishs_r400_win

editing all the tests for being able to run any number of cases

Change 117844 on 2003/08/22 by kevino@kevino_r400_linux_marlboro

Removed some redundant test cases and relabled interlaced formats to INTERLACED instead od INTERLACE.

Change 117841 on 2003/08/22 by ashishs@fl_ashishs_r400_win

edited the test so that it could be run for any number of cases. For ease in Hardware validation

Change 117821 on 2003/08/22 by tmartin@tmartin_r400_win

made the test a little clearer by changing colors and adding better comments

Change 117808 on 2003/08/22 by tmartin@tmartin_r400_win

tests clamp_disable

Change 117807 on 2003/08/22 by tmartin@tmartin_r400_win

updated to test negative clamping

Change 117793 on 2003/08/22 by mkelly@fl_mkelly_r400_win_laptop

Remove JSS state control.

Change 117783 on 2003/08/22 by ashishs@fl_ashishs_r400_win

adding 2 mova tests which were not there in the test_list somehow

Change 117777 on 2003/08/22 by tmartin@tmartin_r400_win

r400vc_base_addr_range_pci_02
r400vc_stride_size_01
r400vc_stride_size_02
r400vc_stride_size_02

Change 117774 on 2003/08/22 by csampayo@fl_csampayo_r400

Add tests r400sq_const_index_05, _06 to tests list and tracker

Change 117766 on 2003/08/22 by csampayo@fl_csampayo2_r400

Updated shader functionality, constant setup/data and tests description accordingly

Change 117756 on 2003/08/22 by kevino@kevino_r400_win_marlboro

Added test case files for tp_multitexture based on fmt # instead of name

Change 117747 on 2003/08/22 by tmartin@tmartin_r400_win

test power of 2 stride values

Change 117735 on 2003/08/22 by ashishs@fl_ashishs_r400_win

edited the test so that it could be run for each single case seperately, easier for hardware validation

Change 117719 on 2003/08/22 by mkelly@fl_mkelly_r400_win_laptop

Remove test examples, check in update to one test....

Change 117713 on 2003/08/22 by tmartin@tmartin_r400_win

added clamping at the end of pci space as an extra check

Change 117675 on 2003/08/21 by ashishs@fl_ashishs_r400_win2

adding Chris Hammer to regressions

Change 117658 on 2003/08/21 by csampayo@fl_csampayo2_r400

Updated to better fulfill test intentions

Change 117642 on 2003/08/21 by tmartin@tmartin_r400_win

added r400vc_array_size_02

Change 117638 on 2003/08/21 by tmartin@tmartin_r400_win

added and removed some comments

Change 117637 on 2003/08/21 by tmartin@tmartin_r400_win

Sets the vertex fetch size constant to 0 and set the distance between buffers 00 and 01 to greater than the max allowed size. If the triangles are drawn this verifies that the size constant value of 0 effectively disables clamping and allows an extra large address to be calculated

Change 117628 on 2003/08/21 by ashishs@fl_ashishs_r400_win

adding Brian Buchner to regressions

Change 117617 on 2003/08/21 by tmartin@tmartin_r400_win

added r400vc_index_rounding_01

Change 117582 on 2003/08/21 by tmartin@tmartin_r400_win

updated now that spec fields have been reversed.

Change 117576 on 2003/08/21 by jhoule@jhoule_r400_linux_marlboro

Testbench now does invalid fetches

Change 117469 on 2003/08/21 by ashishs@fl_ashishs_r400_win

adding Todd to nightly regressions

Change 117382 on 2003/08/20 by csampayo@fl_csampayo_r400

Initial checkin

Change 117381 on 2003/08/20 by csampayo@fl_csampayo_r400

Add test r400sq_const_index_06

Change 117373 on 2003/08/20 by csampayo@fl_csampayo2_r400

Correct vfetches when using FMT_32_FLOAT

Change 117364 on 2003/08/20 by tmartin@tmartin_r400_win

added swizzling tests

Change 117347 on 2003/08/20 by tmartin@tmartin_r400_win

DST_SEL_X sources are SRC_X, SRC_W
DST_SEL_Y sources are SRC_X, SRC_Y
DST_SEL_Z sources are SRC_Y, SRC_Z, 0
DST_SEL_W sources are SRC_Z, SRC_W, 1

Change 117345 on 2003/08/20 by tmartin@tmartin_r400_win

Tests swizzling.
DST_SEL_X sources are SRC_X, 0, 1, Mask
DST_SEL_Y sources are SRC_Y, 0, 1, Mask
DST_SEL_Z sources are SRC_Z, 0, 1, Mask

DST_SEL_W sources are SRC_W, 0, 1, Mask

Change 117344 on 2003/08/20 by tmartin@tmartin_r400_win

Tests swizzling.
DST_SEL_X sources are SRC_Y, SRC_Z
DST_SEL_Y sources are SRC_Z, SRC_W
DST_SEL_Z sources are SRC_X, SRC_W, 0
DST_SEL_W sources are SRC_X, SRC_Y, 1

Change 117339 on 2003/08/20 by csampayo@fl_csampayo2_r400

Correct shaders for proper swizzle of FMT_32 and FM_32_FLOAT vfetches

Change 117319 on 2003/08/20 by csampayo@fl_csampayo2_r400

Revise test to fit its original intentions.

Change 117313 on 2003/08/20 by tmartin@tmartin_r400_win

changed to only use x as a write modifier when 32_float is being used.

Change 117276 on 2003/08/20 by tmartin@tmartin_r400_win

removed some unused frame buffer so the test draws fewer pixels

Change 117227 on 2003/08/20 by jhoule@jhoule_r400_linux_marlboro

Added ch field to the input of the hicolor block.

Change 117223 on 2003/08/20 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint 4, added expected output description.

Change 117222 on 2003/08/20 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint 3, added initial test description.

Change 117220 on 2003/08/20 by mkelly@fl_mkelly_r400_win_laptop

2nd checkpoint, reduced test to 1 case, set all ALU store values to
zero except for the clamp constant K0.

Change 117219 on 2003/08/20 by tmartin@tmartin_r400_win

Removed a swizzle which broke the test when a bug in the emulator was fixed. This way the test is less likely to fail for non-targeted reasons.

Change 117214 on 2003/08/20 by mkelly@fl_mkelly_r400_win_laptop

Initial checkpoint for testing closely knit ALU constant addressing and indexing
throughout the 512 ALU constant store.

Change 117161 on 2003/08/19 by tmartin@tmartin_r400_win

added r400vc_fetch_addr_range_01

Change 117141 on 2003/08/19 by jhoule@jhoule_r400_win_lt

Missing test

Change 117127 on 2003/08/19 by ashishs@fl_ashishs_r400_win

adding new tests

Change 117124 on 2003/08/19 by tmartin@tmartin_r400_win

changed setAddressRange to Set_Size to reflect the change in primlib

Change 117107 on 2003/08/19 by ashishs@fl_ashishs_r400_win

adding test for log_clamp instruction. Also verifying whether log_ieee and log_clamp have the same precision

Change 117089 on 2003/08/19 by ashishs@fl_ashishs_r400_win

testing precision for recip_clamp and recip_ff instructions and verifying it to be the same as recipsq_ieee in the good data range

Change 117077 on 2003/08/19 by ashishs@fl_ashishs_r400_win

adding tests for precision of recip_clamp and recip_ff and verifying whether the precision for both is the same as recip_ieee instruction in the good data range

Change 117061 on 2003/08/19 by mkelly@fl_mkelly_r400_win_laptop

Remove JSS tests..

Change 117059 on 2003/08/19 by mkelly@fl_mkelly_r400_win_laptop

Remove JSS from regress_e, back out this changelist to re-introduce JSS if ever necessary.

Change 117038 on 2003/08/19 by tmartin@tmartin_r400_win

test complete

Change 117037 on 2003/08/19 by tmartin@tmartin_r400_win

test complete

Change 116971 on 2003/08/18 by cbrennan@cbrennan_r400_emu

Shrunk max prim size in effort to reduce the maximum runtime of the test.

Change 116962 on 2003/08/18 by georgev@devel_georgev_r400_lin2_marlboro_tott

More border tests for different formats, adding formats, and changes to regressions.

Change 116879 on 2003/08/18 by tmartin@tmartin_r400_win

Temporarily modified. The top triangle is rgba and the bottom triangle is bgra. They should not be equal, but they are.

Change 116797 on 2003/08/15 by ashishs@fl_ashishs_r400_win2

cheking precision for fract instruction.
TBD : if scalar precision and vector precision need to be checked seperately or any one can be used for precision.

Change 116790 on 2003/08/15 by ashishs@fl_ashishs_r400_win2

updated

Change 116785 on 2003/08/15 by tmartin@tmartin_r400_win

added r400vc_fetch_mode_01 and r400vc_array_size_01

Change 116780 on 2003/08/15 by tmartin@tmartin_r400_win

increased NUM_CASES to 4

Change 116778 on 2003/08/15 by tmartin@tmartin_r400_win

support for FMT_32_32_32_FLOAT was added to primlib so the second case was enabled in this test.

Change 116776 on 2003/08/15 by tmartin@tmartin_r400_win

Sets the maximum value in the vertex fetch constant size field and puts vertex data up to the boundary.

Change 116743 on 2003/08/15 by georgev@devel_georgev_r400_lin2_marlboro_tott

Added split triangle.

Change 116632 on 2003/08/14 by jhoule@jhoule_r400_win_lt

Forgot to add shader program for FMT_32_32_32_FLOAT vfetches

Change 116624 on 2003/08/14 by ashishs@fl_ashishs_r400_win

adding ADD_CONST test. Currently ADD_CONST_0,ADD_CONST_1, scalar ADD all have the same execution code. Also there is a problem with ADD_CONST since it doesn't pick up the right value for the SrcGPR and just initialises to 1

Change 116622 on 2003/08/14 by mkelly@fl_mkelly_r400_win_laptop

Update comment in _11
Copy _11 to _12 and use 144 vertices per packet
Update test_list and tracker accordingly.

Change 116616 on 2003/08/14 by jhoule@jhoule_r400_win_lt

Added support for FMT_32_32_32_FLOAT

Change 116570 on 2003/08/14 by ashishs@fl_ashishs_r400_win

adding new tests

Change 116566 on 2003/08/14 by tmartin@tmartin_r400_win

small changes to aid in figuring out the swizzle bug with floats

Change 116416 on 2003/08/13 by ashishs@fl_ashishs_r400_win

adding test having generalised settings for rendering number of verts per packet as well as controlling number of packets. Also currently having a problem in which no. of vertices > 43 create a failure for the shader logic

Change 116400 on 2003/08/13 by jhoule@jhoule_r400_win_marlboro

Old pixel shaders never submitted

Change 116365 on 2003/08/13 by omesh@omesh_r400_linux_marlboro_tott

Added a few additional testcases to limit the number of registers tested.

Change 116364 on 2003/08/13 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 116362 on 2003/08/13 by cbrennan@cbrennan_r400_release

Forced on mipmapping to hopefully make tests take less time.

Change 116358 on 2003/08/13 by mkelly@fl_mkelly_r400_win_laptop

Add

Change 116357 on 2003/08/13 by mkelly@fl_mkelly_r400_win_laptop

Final, test combinations of addressing and indexing the ALU constant store with coissue and single issue instructions.

Change 116319 on 2003/08/13 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 116194 on 2003/08/12 by tmartin@tmartin_r400_win

added r400vc_endian_swap_01 and r400vc_endian_swap_02

Change 116184 on 2003/08/12 by tmartin@tmartin_r400_win

updated to test bottom of address space and added more triangles.

Change 116174 on 2003/08/12 by tmartin@tmartin_r400_win

Tests swizzling during vfetch instructions.

Change 116173 on 2003/08/12 by tmartin@tmartin_r400_win

Tests index rounding with vfetch instructions.

Change 116172 on 2003/08/12 by tmartin@tmartin_r400_win

Tests the VC's dynamic range when addressing vertex buffers stored in PCI memory. This test shows the upper end of the address range is accessible through PCI.

Change 116171 on 2003/08/12 by tmartin@tmartin_r400_win

Tests endian swap modes 2 and 3.

Change 116170 on 2003/08/12 by tmartin@tmartin_r400_win

Tests endian swap modes 0 and 1.

Change 116157 on 2003/08/12 by omesh@omesh_r400_linux_marlboro_tott

Added another testcase that fast clears, normal clears then again fast
clears to verify h/w synchronization (2nd fast clear should wait for the
normal clear to finish). Ran it on the emulator but haven't run it on
h/w yet.

Change 116139 on 2003/08/12 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 116122 on 2003/08/12 by omesh@omesh_r400_linux_marlboro_tott

Disabled multisampling for 1 sample testcases. This fixes emulator
assertions.

Change 116112 on 2003/08/12 by omesh@omesh_r400_linux_marlboro_tott

Overlooked adding the FB base start address to color0 buffer base address. It should be fine now.

Change 116109 on 2003/08/12 by jhoule@jhoule_r400_linux_marlboro

Reverting old Toronto Makefile changes which weren't undone here

Change 116108 on 2003/08/12 by mkelly@fl_mkelly_r400_win_laptop

Add more description in shaders, fix vertex constant on a few cases...

Change 116072 on 2003/08/12 by csampayo@fl_csampayo2_r400

Initialize unused R1 components

Change 116022 on 2003/08/12 by ashishs@fl_ashishs_r400_win

updating the test. Shaders with different pred instructions setting the pred reg and clr instruction clearing out the register as well as checking MAX_FLOAT

Change 115943 on 2003/08/11 by omesh@omesh_r400_linux_marlboro_tott

Overlooked adding the FB base start address to color0 buffer base address. It should be fine now.

Change 115922 on 2003/08/11 by ashishs@fl_ashishs_r400_win

initial checkin for pred_set_clr instruction. Still doubts regarding the position of this instruction relative to other pred instructions inside shader

Change 115907 on 2003/08/11 by omesh@omesh_r400_linux_marlboro_tott

Added indirect register R/W testing and I may have used different data structures to access register objects. These tests
don't all pass for various reasons: SOme bits are not testable so I have yet to mask r/w operations on them.

Change 115904 on 2003/08/11 by cbrennan@cbrennan_r400_win_marlboro

Added DXT3A, DXT5A and CTX1 to testcases and added random_level4 which includes these.

Change 115871 on 2003/08/11 by ashishs@fl_ashishs_r400_win

test for scalar min instruction

Change 115803 on 2003/08/11 by ashishs@fl_ashishs_r400_win

test for max scalar instruction, testing on xyzw channels with different combination of data

Change 115676 on 2003/08/08 by ashishs@fl_ashishs_r400_win

wasnt using the input vertex for constant indexing

Change 115674 on 2003/08/08 by ashishs@fl_ashishs_r400_win

wasnt correctly using the incoming vertex z value for constant indexing

Change 115673 on 2003/08/08 by ashishs@fl_ashishs_r400_win

adding 2 more tests

Change 115672 on 2003/08/08 by ashishs@fl_ashishs_r400_win

adding test for MAX4 instruction. testing on all pipes as well as input vertex selecting predicates from shader therby selecting constant combination

Change 115644 on 2003/08/08 by ashishs@fl_ashishs_r400_win

had incorrect name inside the test_list

Change 115642 on 2003/08/08 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 115641 on 2003/08/08 by mkelly@fl_mkelly_r400_win_laptop

Update but still asserts

Change 115618 on 2003/08/08 by mkelly@fl_mkelly_r400_win_laptop

Disable MSAA when number of samples is zero. This allows test to finish and not
assert in RB on depth number of samples being zero when msaa is enabled.

Change 115608 on 2003/08/08 by mkelly@fl_mkelly_r400_win_laptop

Add const_index tests...

Change 115607 on 2003/08/08 by mkelly@fl_mkelly_r400_win_laptop

Negative ALU VS constant clamping, negative index clamping with negative stepping

Change 115606 on 2003/08/08 by ashishs@fl_ashishs_r400_win

adding test for DOT2ADD

Change 115587 on 2003/08/08 by mkelly@fl_mkelly_r400_win_laptop

Positive constant clamping with negative loop stepping...

Change 115576 on 2003/08/08 by omesh@omesh_r400_linux_marlboro_tott

Added missing depth clear register programming.

Change 115562 on 2003/08/08 by mkelly@fl_mkelly_r400_win_laptop

ALU positive constant index clamping and constant clamping

Change 115527 on 2003/08/08 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint VS positive constant indexing and clamping

Change 115503 on 2003/08/07 by georgev@devel_georgev_r400_lin2_marlboro_tott

Added stress test to make big objects to fill queues.

Change 115480 on 2003/08/07 by mkelly@fl_mkelly_r400_win_laptop

First test of series which checks positive alu constant index clamping.

Change 115428 on 2003/08/07 by ashishs@fl_ashishs_r400_win2

adding shaders , one with predicates and sub_prev inserted in an particular order to see the effect and in the shader_004 adding another instruction eg. mova in between the predicate instructions to see the effects and the results have been described in the shaders.

Change 115387 on 2003/08/07 by ashishs@fl_ashishs_r400_win2

adding 10 more tests done during last week and the current week

Change 115384 on 2003/08/07 by ashishs@fl_ashishs_r400_win2

adding 4 shaders to cover the SUB instruction on all channels

Change 115372 on 2003/08/07 by csampayo@fl_csampayo2_r400

Extended exports to span 2 framebuffer tiles.

Change 115368 on 2003/08/07 by ashishs@fl_ashishs_r400_win2

performed a test according to Carlos's spec to test if the add and sub instruction yeild any difference/error

Change 115331 on 2003/08/07 by llefebvr@llefebvr_r400_emu_montreal

Conditions were not set correctly (they were swapped).

Change 115328 on 2003/08/07 by llefebvr@llefebvr_r400_emu_montreal

The HW has a 26 bits normilizer so I increased the emulator's precision to match. This fixes r400sq_ripple_01.cpp. I had to re-goldenize 1 test in the regress_e suite.

Change 115304 on 2003/08/06 by csampayo@fl_csampayo2_r400

Fixed color ramp, increased export aperture.

Change 115277 on 2003/08/06 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint, temp save of worksheet as well...

Change 115267 on 2003/08/06 by smoss@smoss_crayola_linux_orl

changed to initialize gpr channels

Change 115266 on 2003/08/06 by ashishs@fl_ashishs_r400_win

initial checkin for testing sub_prev instruction. Seems to have a precision issue and part of shaders 3 and 4 commented out for now.

Change 115226 on 2003/08/06 by ashishs@fl_ashishs_r400_win

didnt have the correct shaders

Change 115220 on 2003/08/06 by ashishs@fl_ashishs_r400_win

adding test for mul_prev with various combination of instructions

Change 115188 on 2003/08/06 by ashishs@fl_ashishs_r400_win

updating the golden.lst file

Change 115187 on 2003/08/06 by ashishs@fl_ashishs_r400_win

updated the image since with the NEW TP using 'setenv TP_V2 3' as suggested by Jocelyn the test produced a different image compared with old environment. But with pix center as 0 (after test was updated) it produces the same image in old as well as new environment and keeping all other same.

Change 115178 on 2003/08/06 by ashishs@fl_ashishs_r400_win

changing the pixel centering, since with the NEW TP using 'setenv TP_V2 3' as suggested by Jocelyn the test produced a different image compared with old environment. But with pix center as 0 it produces the same image in old as well as new and keeping all other same.

Change 115063 on 2003/08/05 by mkelly@fl_mkelly_r400_win_laptop

Check in to save, not complete

Change 115038 on 2003/08/05 by ashishs@fl_ashishs_r400_win

adding test for add_prev instruction with different combination of instructions used to get result in the previousScalar

Change 114967 on 2003/08/05 by ashishs@fl_ashishs_r400_win

modifying the script since it was using different search strings in different areas of the script for the same cause

Change 114947 on 2003/08/05 by mkelly@fl_mkelly_r400_win_laptop

Change HOS reuse depth

Change 114930 on 2003/08/05 by jhoule@jhoule_r400_linux_marlboro

Modified tests to set CLAMP_DISABLE bit (second dword, bit 30), since they are setting a size of 0.

Change 114913 on 2003/08/05 by ashishs@fl_ashishs_r400_win

forgot to update the test for Write_To_Memory function change

Change 114912 on 2003/08/05 by mkelly@fl_mkelly_r400_win_laptop

Modify to work with non-zero FB start

Change 114905 on 2003/08/05 by smoss@smoss_crayola_linux_orl

comment out jss* tests - no longer needed
comment out pipe disable - hardware will change

Change 114832 on 2003/08/04 by csampayo@fl_csampayo3

Update for Write_To_Memory change,

Change 114825 on 2003/08/04 by cbrennan@cbrennan_r400_emu

Removed clamp mode from tests since TP_V2 is enabled by default.

Change 114813 on 2003/08/04 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 114810 on 2003/08/04 by mkelly@fl_mkelly_r400_win_laptop

Simple triangle with SET_STATE

Change 114808 on 2003/08/04 by omesh@omesh_r400_linux_marlboro_tott

Corrected some typos that were mixing up testcase names. Also made some
changes that make these tests run without compiler errors on Windows.

Change 114805 on 2003/08/04 by ashishs@fl_ashishs_r400_win

adding test for killone instruction checking 1 as well as < and > 1 over all the channels using different shaders as well as over all shader pipes

Change 114791 on 2003/08/04 by lseiler@lseiler_r400_win_marlboro

Sets msaa_enable only with >1 sample

Change 114778 on 2003/08/04 by ashishs@fl_ashishs_r400_win

adding test for pred_set_restore instruction

Change 114768 on 2003/08/04 by ashishs@fl_ashishs_r400_win

adding test for pred_set_pop instruction using 4 shaders to tests each x,y,z,w channel with different data inputs wrt (> = <) 0

Change 114764 on 2003/08/04 by mkelly@fl_mkelly_r400_win_laptop

Test Mainline

Change 114757 on 2003/08/04 by ashishs@fl_ashishs_r400_win

Test had a problem while running hardware since the c31 which was used in pixel shader wasnt initialised explicitly, and in emulator it gets initialised to 0's explicitly

Change 114743 on 2003/08/04 by mkelly@fl_mkelly_r400_win_laptop

Branch

Change 114715 on 2003/08/04 by smoss@smoss_crayola_linux_orl

makefile for vc

Change 114577 on 2003/08/01 by ashishs@fl_ashishs_r400_win

adding test for pred_set_inv. each shader checks unique combination of (+,-,0,1) for x,y,z,w channels

Change 114574 on 2003/08/01 by georgev@devel_georgev_r400_lin2_marlboro_tott

Partial complete test.

Change 114560 on 2003/08/01 by ashishs@fl_ashishs_r400_win

corrected an error where the script was counting VC tests as SP tests

Change 114549 on 2003/08/01 by tmartin@tmartin_r400_win

added r400vc_fetch_mode_02

Change 114543 on 2003/08/01 by tmartin@tmartin_r400_win

Tests mega fetch counts of 0 thru 3.

Change 114542 on 2003/08/01 by tmartin@tmartin_r400_win

Tests mega fetch counts of 4 - 7. Including 1 - 4 dword mini fetches.

Change 114534 on 2003/08/01 by ashishs@fl_ashishs_r400_win2

added variables for the new changes done in AUTO_..sync.tcsh file

Change 114454 on 2003/08/01 by ashishs@fl_ashishs_r400_win

adding vector scalar and coissue instructions

Change 114428 on 2003/08/01 by ashishs@fl_ashishs_r400_win

adding scalar vector and coissue shaders

Change 114424 on 2003/08/01 by ashishs@fl_ashishs_r400_win

had wrong shaders

Change 114343 on 2003/07/31 by csampayo@fl_csampayo_r400

Add memory export test, update test list and tracker

Change 114282 on 2003/07/31 by tmartin@tmartin_r400_win

added r400vc_addr_spanning_01

Change 114277 on 2003/07/31 by tmartin@tmartin_r400_win

Tests that the VC correctly spans address spaces

Change 114221 on 2003/07/31 by ashishs@fl_ashishs_r400_win

added shaders for scalar, vector and coissue instructions

Change 114192 on 2003/07/31 by tien@tien_r400_devel_marlboro

Bug fixes and change in tpc_out_fifo width

Change 114148 on 2003/07/31 by ashishs@fl_ashishs_r400_win2

adding a new line at the end of test_list since my script needs a newline at the end of test_list. Later would change my script

Change 114080 on 2003/07/30 by tmartin@tmartin_r400_win

fixed the NUM_CASES loop

Change 114055 on 2003/07/30 by ashishs@fl_ashishs_r400_win

added shaders for pred_sete with scalar and coissue instructions

Change 114018 on 2003/07/30 by tmartin@tmartin_r400_win

added strides/offsets tests

Change 114004 on 2003/07/30 by tmartin@tmartin_r400_win

Check fetch address calculations and range (including negative results)

Change 114003 on 2003/07/30 by tmartin@tmartin_r400_win

Tests that the VC correctly clamps addresses that are outside of the constant size.

Change 113997 on 2003/07/30 by tmartin@tmartin_r400_win

test currently works, but is restricted to a hardcoded range in the address space. Eventually this restriction will be removed and the test will be modified to span the entire address range.

Change 113981 on 2003/07/30 by omesh@omesh_r400_linux_marlboro_tott

Fixed a bug of not setting the Z_BASE address correctly (Overlooked adding the FB start address).
Tests now pass, as verified by John Chen.

Change 113962 on 2003/07/30 by ashishs@fl_ashishs_r400_win

changed the case for "TOTAL" since Joe's script on web was searching for case sensitive "TOTAL"

Change 113896 on 2003/07/29 by ashishs@fl_ashishs_r400_win

tried to get the automatic submit to P4 to work ....but somehow there are issues related to submit and makes it very complicated to do automatic submit.

Change 113874 on 2003/07/29 by ashishs@fl_ashishs_r400_win

increased the length of test names from 50 to 80

Change 113871 on 2003/07/29 by tmartin@tmartin_r400_win

updated to stress all 64 vertex pipes

Change 113869 on 2003/07/29 by ashishs@fl_ashishs_r400_win

adding option "p"
if the script is run with -p then it will update the web as well as tracker
else it will just update the tracker

Change 113864 on 2003/07/29 by ashishs@fl_ashishs_r400_win

updated so that the script could be run from any directory. Also corrected a minor error related to the VC page. Also integrated the web script with this script.

Change 113855 on 2003/07/29 by ashishs@fl_ashishs_r400_win

updated the changes for running from any directory

Change 113850 on 2003/07/29 by ashishs@fl_ashishs_r400_win

removing 7 files that were duplicated

Change 113810 on 2003/07/29 by tmartin@tmartin_r400_win

added more vertices to the scenes to saturate all 64 vertex pipes

Change 113809 on 2003/07/29 by omesh@omesh_r400_linux_marlboro_tott

Made index buffer code less sloppy.

Change 113795 on 2003/07/29 by omesh@omesh_r400_linux_marlboro_tott

Added a testcase "tri3_8x8_replace_8samp" that Bill needed. Ran it and
visually inspected/verified the FB dump.

Change 113706 on 2003/07/28 by ashishs@fl_ashishs_r400_win

updating a minor error in update tracker script. And formatting the update web script to have the test number and serial number as well as block name in the report. Also formatting the output.

Change 113629 on 2003/07/28 by ashishs@fl_ashishs_r400_win

corrected minor error while printing

Change 113627 on 2003/07/28 by ashishs@fl_ashishs_r400_win

Upadte for script to update regression tracker
1. Maintain a seperate worksheet for each emulator block
2. Add numbering for each page/block
3. Updated paths for the tracker viz put the tracker in t:\r400\ making it available for updating to others

Change 113623 on 2003/07/28 by tien@tien_r400_devel_marlboro

Man it's been a long time coming :-)
formatter fix for TP to output to 1 simd only
drive simd signal from TPC to VC (will prolly need to skew it a bit, but that will fall out from debug)
Clean up get/set logic

Change 113606 on 2003/07/28 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 113594 on 2003/07/28 by mkelly@fl_mkelly_r400_win_laptop

Validate upper SIMD1/0 bits for pipe disable

Change 113593 on 2003/07/28 by ashishs@fl_ashishs_r400_win

adding Todd to email list

Change 113470 on 2003/07/28 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 113444 on 2003/07/27 by ashishs@fl_ashishs_r400_win

adding blocks CP,RBBM,VC,BUGS,SANITY,STRESS,PERF

Change 113432 on 2003/07/27 by ashishs@fl_ashishs_r400_win

adding option "i" using which the regress_r400 script will use the hardcoded paths for each of the test

Change 113284 on 2003/07/25 by tmartin@tmartin_r400_win

added tests

Change 113282 on 2003/07/25 by csampayo@fl_csampayo_r400

Updated status for tests r400sx_vtx_point_size_export_01-04 and added them to test_list

Change 113264 on 2003/07/25 by mkelly@fl_mkelly_r400_win_laptop

update...

Change 113261 on 2003/07/25 by mkelly@fl_mkelly_r400_win_laptop

Update Tess level, Hos reuse, dealloc distance, reduce fb size

Change 113226 on 2003/07/25 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 113218 on 2003/07/25 by ashishs@fl_ashishs_r400_win

changing names for output files with time stamp

Change 113206 on 2003/07/25 by tmartin@tmartin_r400_win

Tests the VC's stride range when addressing vertex buffers stored in the frame buffer. Stride range is 0:255.

Change 113179 on 2003/07/25 by ashishs@fl_ashishs_r400_win

corrected the time formatting in output

Change 113176 on 2003/07/25 by ashishs@fl_ashishs_r400_win

updated to create the statistics page based on the tracker

Change 113154 on 2003/07/25 by csampayo@fl_csampayo2_r400

Correct test _01 to use IM_LOAD for loading shaders and, add 3 new test with different patterns

Change 113145 on 2003/07/25 by ashishs@fl_ashishs_r400_win

merging the statistics data generated from report with the tracker also sorting records with test status (PASS/FAIL)

Change 113126 on 2003/07/25 by mkelly@fl_mkelly_r400_win_laptop

Validate new state bit, SC_LINE_CNTL.LAST_PIXEL behaviour

Change 113071 on 2003/07/24 by csampayo@fl_csampayo2_r400

Update for non-zero FB start wrt depth buffer setup

Change 113023 on 2003/07/24 by ashishs@fl_ashishs_r400_win

initial checkin for script which will get the TRACKER data and update it on the web

Change 112997 on 2003/07/24 by ashishs@fl_ashishs_r400_win

using a different way to output runTimes in the Excel

Change 112984 on 2003/07/24 by llefebvr@llefebvre_laptop_r400_emu

Fixing test that wasn't initializing all GPR fields before exporting.

Change 112980 on 2003/07/24 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 112966 on 2003/07/24 by mkelly@fl_mkelly_r400_win_laptop

Extensive round mode testing...

Change 112965 on 2003/07/24 by smoss@smoss_crayola_linux_orl

removed die, added gold

Change 112950 on 2003/07/24 by omesh@omesh_r400_linux_marlboro_tott

Again, modified the condition under which the Tile Buffer is needed:
When Stencil is Fast Expanded, this is not reflected in the Depth
Buffer, it is only done in the Tile Buffer, so even when Stencil Expand is
done, this means the Depth Buffer is not Fully Expanded and hence the
Tile Buffer is needed.

Change 112946 on 2003/07/24 by omesh@omesh_r400_linux_marlboro_tott

Added an additional condition under which the Tile Buffer needs to be

dumped (When Z is NOT Fast expanded but Stencil IS, the actual stencil
value does not get written to the Depth Buffer and hence the tile buffer
is needed)

Change 112925 on 2003/07/24 by omesh@omesh_r400_linux_marlboro_tott

Added Depth Clear register programming. John/Paul, please verify that
this problem is now fixed.

Change 112916 on 2003/07/24 by cbrennan@cbrennan_r400_emu

Added a check for system memory allocation to tests so prevent malloc failing even when it would compress into the frame buffer just fine.

Change 112898 on 2003/07/24 by ashishs@fl_ashishs_r400_win

initial checkin for script which will update a centralised tracker located at TBD based on each one's regression run

Change 112888 on 2003/07/24 by mkelly@fl_mkelly_r400_win_laptop

PA_SU_VTX_CNTL Round and Quantization variations...

Change 112821 on 2003/07/23 by ashishs@fl_ashishs_r400_win

corrected the scaling on the tests to match r390

Change 112805 on 2003/07/23 by ashishs@fl_ashishs_r400_win

updating the test_list as according to all other test lists with a newline at the end for now. Later on I will change my script to handle this.

Change 112792 on 2003/07/23 by ashishs@fl_ashishs_r400_win

updating the newly added SIN / COS tests

Change 112788 on 2003/07/23 by ashishs@fl_ashishs_r400_win

adding more sin tests after converting from r390

Change 112784 on 2003/07/23 by csampayo@fl_csampayo_r400

Forgot to save updates

Change 112781 on 2003/07/23 by csampayo@fl_csampayo_r400

Add r400vgt_vtx_export_very_very_simple_05 to _10 tests, sort list

Change 112778 on 2003/07/23 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 112777 on 2003/07/23 by ashishs@fl_ashishs_r400_win

adding tests converted from r390

Change 112771 on 2003/07/23 by tmartin@tmartin_r400_win

Tests the VC's dynamic range when addressing vertex buffers stored in PCI memory.

Change 112764 on 2003/07/23 by ashishs@fl_ashishs_r400_win

changed the frame buf background color to match r390

Change 112761 on 2003/07/23 by csampayo@fl_csampayo2_r400

Updated for FB start and added missing vtx I and pix I shaders

Change 112714 on 2003/07/23 by ashishs@fl_ashishs_r400_win

adding test for newly added COS opcode. (converted from r390 Carlos's test)

Change 112707 on 2003/07/23 by ashishs@fl_ashishs_r400_win

Adding test for newly added SIN opcode in emulator (converted from Carlos's r390 test)

Change 112685 on 2003/07/23 by cbrennan@cbrennan_r400_release

Fixed test so that cubic dimension was properly loaded into constant before calling getAllocationSizeInBytes.

Change 112675 on 2003/07/23 by rmanapat@rmanapat_r400_sun_marlboro

Fix should reduce how long the test runs

Change 112654 on 2003/07/23 by csampayo@fl_csampayo2_r400

Minimum number of cases that cause problem

Change 112634 on 2003/07/23 by mkelly@fl_mkelly_r400_win_laptop

VC is always on, always use variable FB start (non-zero).

Change 112615 on 2003/07/23 by ashishs@fl_ashishs_r400_win

Regress r400 script Update :
1. In the Summary report add Regression Start Time and Regression End time .

2. In the Summary report add Total Emulator Elapsed Time for Each Block. So Each block will have the time it used for regressions.
3. Also in "s" option for the regress_r400 script now it doesn't search for the sync. It will accept the sync_number from parameter command line and then stamp it on the output directory. Also the code for finding the sync_number, still exists but the "s" option will no longer use that code.
4. Option "p" for publishing to the web will work the same way it has been working and the code has not been touched.

Change 112598 on 2003/07/23 by smoss@smoss_crayola_linux_orl

creating golds for one test

Change 112523 on 2003/07/22 by csampayo@fl_csampayo_r400

Add flush before dumping image

Change 112517 on 2003/07/22 by tmartin@tmartin_r400_win

Tests the VC's address range when addressing vertex buffers stored in AGP memory.

Change 112516 on 2003/07/22 by csampayo@fl_csampayo_r400

Add flush before dumping image

Change 112501 on 2003/07/22 by csampayo@fl_csampayo2_r400

Update for non-zero FB start

Change 112476 on 2003/07/22 by mkelly@fl_mkelly_r400_win_laptop

Invalidate VC

Change 112461 on 2003/07/22 by mkelly@fl_mkelly_r400_win_laptop

Quant mode stepping postion in 1/32 steps, varying quantization

Change 112457 on 2003/07/22 by csampayo@fl_csampayo2_r400

Update for non-zero FB start

Change 112426 on 2003/07/22 by mkelly@fl_mkelly_r400_win_laptop

Validate quant modes 1/16, 1/8, 1/4, 1/2, 1

Change 112409 on 2003/07/22 by ashishs@fl_ashishs_r400_win

added test for setgt instruction with scalar vector and coissue instructions

Change 112380 on 2003/07/22 by csampayo@fl_csampayo2_r400

Updated pattern for testing

Change 112356 on 2003/07/22 by jyarasca@jyarasca_r400_win_cvd

Makefile changes: WLOPTS, ULOPTS, LFLAGS_WIN, and LFLAGS_SUN variables need only be assigned to $(DEPTH)/auto/bin
now since .dll and .lib are all copied to that directory

Change 112350 on 2003/07/22 by ashishs@fl_ashishs_r400_win

finalising test

Change 112342 on 2003/07/22 by ashishs@fl_ashishs_r400_win2

initial checkin

Change 112339 on 2003/07/22 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 112332 on 2003/07/22 by llefebvr@llefebvre_laptop_r400_emu

I had the old output order for cube: ma,faceid,sc,tc. I changed it for the new one: tc,sc,ma,faceid.

Change 112322 on 2003/07/22 by mkelly@fl_mkelly_r400_win_laptop

Add sx_rb.dmp compare for a test, modify regress script to handle it

Change 112312 on 2003/07/22 by jayw@jayw_r400_linux_marlboro

Hooking up register read logic. removed dummy memory from rb_rbt_fragment_fifo.v

Change 112254 on 2003/07/21 by llefebvr@llefebvre_laptop_r400_emu

This is the CUBE opcode change that takes into account the recent HW change. I also modified one test case that was wrongfully picking W as the FACEID (it is Y).

Change 112249 on 2003/07/21 by ashishs@fl_ashishs_r400_win2

updating shaders and test

Change 112226 on 2003/07/21 by tmartin@tmartin_r400_win

Tests the VC's dynamic range when addressing vertex buffers stored in a 32 MB frame buffer.

Change 112225 on 2003/07/21 by tmartin@tmartin_r400_win

Tests the VC's dynamic range when addressing vertex buffers stored in a 64 MB frame buffer.

Change 112224 on 2003/07/21 by tmartin@tmartin_r400_win

Tests the VC's dynamic range when addressing vertex buffers stored in a 128 MB frame buffer.

Change 112223 on 2003/07/21 by tmartin@tmartin_r400_win

Tests the VC's dynamic range when addressing vertex buffers stored in a 256 MB frame buffer.

Change 112221 on 2003/07/21 by tmartin@tmartin_r400_win

Tests the VC's dynamic range when addressing vertex buffers stored in the frame buffer.

Change 112212 on 2003/07/21 by kryan@kryan_r400_win_marlboro_XP

Fix missing ; at end of line in shaders

Change 112197 on 2003/07/21 by mkelly@fl_mkelly_r400_win_laptop

Invalidate vc

Change 112177 on 2003/07/21 by mkelly@fl_mkelly_r400_win_laptop

Updates, including invalidating the VC and hos reuse depth setting...

Change 112133 on 2003/07/21 by mkelly@fl_mkelly_r400_win_laptop

Update HOS reuse...

Change 112119 on 2003/07/21 by mkelly@fl_mkelly_r400_win_laptop

Change vs_export from 4 to 3...

Change 112104 on 2003/07/21 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 112089 on 2003/07/21 by mkelly@fl_mkelly_r400_win_laptop

Prim type detection on RT and non-RT update...

Change 112088 on 2003/07/21 by askende@askende_r400_linux_marlboro updated the cube tests to comply with the new definition of the CUBE instruction

Change 111988 on 2003/07/18 by csampayo@fl_csampayo2_r400

Updated export mode pattern

Change 111979 on 2003/07/18 by ashishs@fl_ashishs_r400_win

initial checkin

Change 111975 on 2003/07/18 by csampayo@fl_csampayo_r400

Adding point size export mode test. Updated test_list and test tracker accordingly.

Change 111955 on 2003/07/18 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 111951 on 2003/07/18 by georgev@devel_georgev_r400_lin2_marlboro_tott

Tests for extra format coverage.

Change 111944 on 2003/07/18 by mkelly@fl_mkelly_r400_win_laptop

Update..

Change 111937 on 2003/07/18 by ashishs@fl_ashishs_r400_win

adding vector, scalar and coissue instructions

Change 111910 on 2003/07/18 by mkelly@fl_mkelly_r400_win_laptop

Revised tests..

Change 111753 on 2003/07/17 by ashishs@fl_ashishs_r400_win

added shaders for vector, scalar and coissue instructions using the same test

Change 111731 on 2003/07/17 by mkelly@fl_mkelly_r400_win_laptop

Converted test to PM4 and support two pixel shaders...

Change 111723 on 2003/07/17 by jayw@jayw_r400_linux_marlboro

Some added tests.

Change 111720 on 2003/07/17 by cbrennan@cbrennan_r400_emu

Added utils to calculate allocation sizes of maps in bytes.

Increased ranges of most tc random tests.

Change 111663 on 2003/07/17 by omesh@omesh_r400_linux_marlboro_only_devel

Since Primlib adds a restriction that surface heights need to be 32 aligned, I had to increase the ZCOUNT address range from 256 bytes to 256*32 bytes. The tests now don't assert, although we are dumping unnecessarily larger chunks of memory. Later, maybe Kevin Ryan could introduce an "overide" bit that doesn't require surface heights to be multiples of 32....

I also fixed a bug in which the correct area of memory was not being identified (Framebuffer base address was not being added into the full device address)

Change 111661 on 2003/07/17 by ashishs@fl_ashishs_r400_win

corrected an error

Change 111658 on 2003/07/17 by ashishs@fl_ashishs_r400_win

updated so that could do 128 points/pixels per packet therby filling up shader pipes

Change 111657 on 2003/07/17 by georgev@devel_georgev_r400_lin2_marlboro_tott

Changed frame_buffer for multicontext texture tests.

Change 111655 on 2003/07/17 by mkelly@fl_mkelly_r400_win_laptop

Added a separate RT shader for unique param_gen scenarios when comparing
RT to non-RT testing..

Change 111654 on 2003/07/17 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint on upgrading this test ...

Change 111636 on 2003/07/17 by ashishs@fl_ashishs_r400_win

added vector scalar and coissue shaders

Change 111591 on 2003/07/17 by mkelly@fl_mkelly_r400_win_laptop

Change vs_eport from 4 to 3 on the non-RTS and modify test to work for gpr destination 4 instead of 5 for the generated parameter. This allows the non-RTS and RTS pixel shader to be shared.

Change 111546 on 2003/07/16 by cbrennan@cbrennan_r400_emu

Add more formats for support to getSize_bits and adjust test to properly deal with DXT size calculations.

Change 111539 on 2003/07/16 by ashishs@fl_ashishs_r400_win

using the same test without changing the output to render 128 pixels/points at a time in a packet thereby filling up shader pipes

Change 111511 on 2003/07/16 by jhoule@jhoule_r400_win_lt

Updated example file with new vtx_ fields.

Change 111500 on 2003/07/16 by mkelly@fl_mkelly_r400_win_laptop

Updated simple reference test to work with variable FB start...

Change 111493 on 2003/07/16 by ashishs@fl_ashishs_r400_win

updated tests same as previous with scalar, vector and coissue shaders

Change 111490 on 2003/07/16 by mkelly@fl_mkelly_r400_win_laptop

Simple Triangle with POS,COLOR,NORMAL, does a VS_DONE_TS event at end of triangle

Change 111459 on 2003/07/16 by ashishs@fl_ashishs_r400_win

added shaders for scalar vector and coissue in the same test

Change 111458 on 2003/07/16 by cbrennan@cbrennan_r400_emu

Increased ranges to 2kx2kx1k while decreasing size limit to 64meg.

Change 111431 on 2003/07/16 by ashishs@fl_ashishs_r400_win

noticed that wasnt using w channel in scalar instructions and updated to use w channel as well

Change 111423 on 2003/07/16 by ashishs@fl_ashishs_r400_win

adding 3 shaders for the test with each shader being the same except they differ as being coissue or vector or scalar shaders therby covering all types of instructions with KILLGT in a single test

Change 111376 on 2003/07/16 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 111375 on 2003/07/16 by mkelly@fl_mkelly_r400_win_laptop

Fix buffers to work with variable FB

Change 111349 on 2003/07/16 by mkelly@fl_mkelly_r400_win_laptop

Update Z_BASE to work with variable FB...

Change 111346 on 2003/07/16 by kevino@kevino_r400_emu

Added validateTexDim() function- tells if tfc dimension is OK for the data format

Change 111334 on 2003/07/16 by ashishs@fl_ashishs_r400_win2

updated tests replacing Write_To_Memory function

Change 111272 on 2003/07/15 by mkelly@fl_mkelly_r400_win_laptop

Fix Z_BASE to work with variable FB

Change 111263 on 2003/07/15 by mkelly@fl_mkelly_r400_win_laptop

Fix Z_BASE for variable FB

Change 111253 on 2003/07/15 by mkelly@fl_mkelly_r400_win_laptop

Fix Z_BASE to work with variable FB start...

Change 111242 on 2003/07/15 by jhoule@jhoule_r400_win_lt

Added support for various other interesting primitives...

Change 111240 on 2003/07/15 by tmartin@tmartin_r400_win

added r400vc_addr_alignment_01

Change 111235 on 2003/07/15 by tmartin@tmartin_r400_win

update

Change 111225 on 2003/07/15 by ashishs@fl_ashishs_r400_win

changed the Write_to_memory function. shouldnt affect test

Change 111212 on 2003/07/15 by ashishs@fl_ashishs_r400_win

replacing Write_To_Memory function with new primlib function. Shouldnt affect test

Change 111210 on 2003/07/15 by kevino@kevino_r400_linux_marlboro

Added 1D and 3D textures.

Change 111209 on 2003/07/15 by mkelly@fl_mkelly_r400_win_laptop

Fix Z_BASE to work with variable FB

Change 111208 on 2003/07/15 by mkelly@fl_mkelly_r400_win_laptop

Add rbbm_vc

Change 111194 on 2003/07/15 by omesh@omesh_r400_linux_marlboro_only_devel

Added RB register Read/Write tests, however, haven't yet incorporated
Read/Write Masks and masks to ignore reserved bits in registers. This
information is not easily extractable from Autoreg even if it does
exist, so I will add in the masks later.
Also, I haven't yet incorporated a tracker (if one exists) for the Host
interface to the registers, or found a way to put a test in
"test determined pass/fail mode".
There are 102 registers which I may trim down later.
The tests currently don't pass, even though the script calls it a
"Pass".

Change 111165 on 2003/07/15 by hartogs@fl_hartogs2

CHanged the HOS_REUSE_DEPTH to reflect the pipe configuration.

Change 111130 on 2003/07/15 by ashishs@fl_ashishs_r400_win

sorting test_list

Change 111126 on 2003/07/15 by ashishs@fl_ashishs_r400_win

added tests

Change 111035 on 2003/07/14 by ashishs@fl_ashishs_r400_win

disabling auto_wrapping_memories test for now

Change 111016 on 2003/07/14 by georgev@devel_georgev_r400_lin2_marlboro_tott

Added 128 bit hi color tests and lists for coverage.

Change 110996 on 2003/07/14 by tmartin@tmartin_r400_win

Tests writing and displaying vertex buffers to all 16 dword alignments in the VC.

Change 110994 on 2003/07/14 by ashishs@fl_ashishs_r400_win

updating the remaining failed SU tests and making sure that the update doesnt affect the failed image

Change 110981 on 2003/07/14 by kevino@kevino_r400_linux_marlboro

tp_perf_2d test but with 4 textures. They use the same coords, format, etc,

Change 110975 on 2003/07/14 by ashishs@fl_ashishs_r400_win2

forgot to update this test for write_to_memory function

Change 110973 on 2003/07/14 by ashishs@fl_ashishs_r400_win

updated test description as well as shaders

Change 110969 on 2003/07/14 by ashishs@fl_ashishs_r400_win

updated test description and shaders

Change 110951 on 2003/07/14 by csampayo@fl_csampayo2_r400

Cleanup cpp, remove co-issue from pix shader, fix vtx shader since, it was using r32 instead of r35 for predicate condition.

Change 110946 on 2003/07/14 by ashishs@fl_ashishs_r400_win

updating test description and shader as well

Change 110931 on 2003/07/14 by ashishs@fl_ashishs_r400_win

removing unwanted code

Change 110929 on 2003/07/14 by ashishs@fl_ashishs_r400_win

updating test description as well as pix shader to output the value produced in the output register

Change 110927 on 2003/07/14 by mkelly@fl_mkelly_r400_win_laptop

Add vc_sq.dmp

Change 110921 on 2003/07/14 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 110911 on 2003/07/14 by mkelly@fl_mkelly_r400_win_laptop

Fix to work with variable FB start...

Change 110901 on 2003/07/14 by ashishs@fl_ashishs_r400_win

using the same test as killge but making sure that that when SRC A and B have equal values the pixel isnt killed...

Change 110891 on 2003/07/14 by ashishs@fl_ashishs_r400_win2

updating tests replacing the new Write_To_Memory function with the new primlib function. Also cleaning out wait_gfx_idle from inside the loop and adding gfx_idle_no_flush if write_to_memory(index_buffer_ptr) present

Change 110856 on 2003/07/14 by ashishs@fl_ashishs_r400_win

carefully selecting a set of pixel to be killed using constants and kill logic in pixel shader

Change 110837 on 2003/07/14 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 110790 on 2003/07/14 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 110685 on 2003/07/12 by ashishs@fl_ashishs_r400_win

updating test to have the same output as the kille using different set of constants than kille test and modifying the pix shader

Change 110600 on 2003/07/11 by ashishs@fl_ashishs_r400_win

initial checkin

Change 110593 on 2003/07/11 by mkelly@fl_mkelly_r400_win_laptop

Fix vs_eport

Change 110592 on 2003/07/11 by mkelly@fl_mkelly_r400_win_laptop

Change VS_EXPORT from 4 to 3 for 4 parameters.

Change 110586 on 2003/07/11 by tmartin@tmartin_r400_win

Vertex shader for testing address dword alignment in the Vertex Cache.

Change 110585 on 2003/07/11 by tmartin@tmartin_r400_win

Pixel shader for testing address dword alignment in the Vertex Cache

Change 110584 on 2003/07/11 by tmartin@tmartin_r400_win

Tests address dword alignment in the Vertex Cache.

Change 110580 on 2003/07/11 by mkelly@fl_mkelly_r400_win_laptop

Update for FB start variable...

Change 110571 on 2003/07/11 by ashishs@fl_ashishs_r400_win

updated such that now the shader kills every other alternate 4th pixel out of 500 points. Each point is mapped as a pixel with the color register containing data to either kill or save the pixel. The 512 consants and the shader have been setup such a way that it kills 3 pixels and outputs every other alternate 4th pixel.

Change 110539 on 2003/07/11 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 110523 on 2003/07/11 by llefebvr@llefebvr_r400_emu_montreal

Fixing shader. Was using address register wihout doing a mova at all!

Change 110515 on 2003/07/11 by mkelly@fl_mkelly_r400_win_laptop

tesselation level ovveride added...

Change 110505 on 2003/07/11 by tien@tien_r400_devel_marlboro

tp_lod_deriv fix (denorms -> 0 in fp_fast_mult_dno)
formatter fix (1.0 detection for 32- and 16- bit channels)

Change 110502 on 2003/07/11 by smoss@smoss_crayola_linux_orl

update

Change 110501 on 2003/07/11 by ashishs@fl_ashishs_r400_win

added tests

Change 110468 on 2003/07/11 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 110466 on 2003/07/11 by jhoule@jhoule_r400_linux_marlboro

Set the execute bit in the permission

Change 110445 on 2003/07/11 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 110433 on 2003/07/11 by ashishs@fl_ashishs_r400_win

adding 4 cases , one for each frustum corner

Change 110430 on 2003/07/11 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 110412 on 2003/07/11 by ashishs@fl_ashishs_r400_win

another line polymode frustum clipping perspective test

Change 110411 on 2003/07/11 by ashishs@fl_ashishs_r400_win2

updated more than 300 CL/VTE tests with a new function that replaces Write_To_Memory function. Also had to get wait_gfx_idle outside loop and no_flush inside loop if write_to_memory index_buffer_ptr

Change 110385 on 2003/07/11 by ashishs@fl_ashishs_r400_win2

added marcos to list

Change 110306 on 2003/07/10 by csampayo@fl_csampayo2_r400

Remove accessing same register in co-issue instruction by removing scalar part

Change 110303 on 2003/07/10 by ashishs@fl_ashishs_r400_win

renaming a test and also adding multiple cases to clip on all the frustum corners

Change 110284 on 2003/07/10 by cbrennan@cbrennan_r400_emu

Refix test.

Change 110277 on 2003/07/10 by mkelly@fl_mkelly_r400_win_laptop

Update..

Change 110276 on 2003/07/10 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 110270 on 2003/07/10 by mkelly@fl_mkelly_r400_win_laptop

Invalidate VC for vertex buffer updating..

Change 110249 on 2003/07/10 by ashishs@fl_ashishs_r400_win

another clip polymode line fill persp test with one edge going thru frustum corner. also each 2 edges having equal w's to each other

Change 110229 on 2003/07/10 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 110208 on 2003/07/10 by mkelly@fl_mkelly_r400_win_laptop

Delete...

Change 110197 on 2003/07/10 by kevino@kevino_r400_win_marlboro

Had to rebuild the other files from the tclist file

Change 110196 on 2003/07/10 by mkelly@fl_mkelly_r400_win_laptop

Updated to work with variable FB start...

Change 110195 on 2003/07/10 by kevino@kevino_r400_win_marlboro

needed to capitalize FLOAT in format

Change 110186 on 2003/07/10 by csampayo@fl_csampayo2_r400

Disable VC caches

Change 110182 on 2003/07/10 by csampayo@fl_csampayo2_r400

Disable VC caches

Change 110171 on 2003/07/10 by kevino@kevino_r400_win_marlboro

added all testcases previously available in fmt8888 to the rest of the existing formats

Change 110169 on 2003/07/10 by mkelly@fl_mkelly_r400_win_laptop

Update to work with variable FB start...

Change 110141 on 2003/07/10 by ashishs@fl_ashishs_r400_win

another polymode line_filled test with diff frustum settings

Change 110139 on 2003/07/10 by ashishs@fl_ashishs_r400_win

another polymode enabled line fill test with different frustum settings

Change 110134 on 2003/07/10 by mkelly@fl_mkelly_r400_win_laptop

Update for variable FB start...

Change 110067 on 2003/07/09 by jhoule@jhoule_r400_win_lt

Removed extension; changed format to 8888 instead of 2101010 (which can't be filtered anyways).

Change 110054 on 2003/07/09 by ashishs@fl_ashishs_r400_win

adding chris to the email_list

Change 110051 on 2003/07/09 by csampayo@fl_csampayo2_r400

Forgot to cleanup this one.

Change 110050 on 2003/07/09 by ashishs@fl_ashishs_r400_win

added test

Change 110048 on 2003/07/09 by csampayo@fl_csampayo2_r400

Update for multiple discrete vertex buffers.

Change 110034 on 2003/07/09 by ashishs@fl_ashishs_r400_win

test carried out according to Mike Mang and Carlos's requirements that show that the clipping still fails when polymode line_fill is enabled with 2 vertices of a triangle having the same w wheras another vertex having a different w.

Change 110001 on 2003/07/09 by llefebvr@llefebvr_r400_emu_montreal

Fixing shader, mova and use of mova separated by a clause boundary.

Change 109999 on 2003/07/09 by cbrennan@cbrennan_r400_emu

Width and height needed to be padded to pitches for randoms.

Change 109972 on 2003/07/09 by mkelly@fl_mkelly_r400_win_laptop

Update for FB

Change 109968 on 2003/07/09 by ashishs@fl_ashishs_r400_win

initial checkin for the kille test

Change 109963 on 2003/07/09 by omesh@omesh_r400_linux_marlboro_only_devel

Framebuffer was being set twice (incorrectly the 2nd time). Fixed the

test.

Change 109937 on 2003/07/09 by mkelly@fl_mkelly_r400_win_laptop

Update to work with any frame buffer address...

Change 109877 on 2003/07/09 by mkelly@fl_mkelly_r400_win_laptop

Update offset from FB

Change 109869 on 2003/07/09 by mkelly@fl_mkelly_r400_win_laptop

Change test to work with variable frame buffer start...

Change 109813 on 2003/07/08 by llefebvr@llefebvr_r400_emu_montreal

Adding a test case to mova_tests.cpp.

Change 109802 on 2003/07/08 by vgoel@fl_vgoel2

fixed for not using uninitialized GPR

Change 109769 on 2003/07/08 by cbrennan@cbrennan_r400_linux_marlboro

Change tests to write unique filenames and delete them when finished. Fixes random seg-faults.

Change 109765 on 2003/07/08 by georgev@devel_georgev_r400_lin2_marlboro_tott

Support of 16 bit textures and coverage for TC.

Change 109734 on 2003/07/08 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 109733 on 2003/07/08 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 109731 on 2003/07/08 by smoss@smoss_crayola_linux_orl

link all subordinate files

Change 109727 on 2003/07/08 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 109722 on 2003/07/08 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 109695 on 2003/07/08 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 109684 on 2003/07/08 by mkelly@fl_mkelly_r400_win_laptop

Make new VB each time the VB is updated, work in progress on a few new tests..

Change 109600 on 2003/07/07 by ashishs@fl_ashishs_r400_win

adding PRED_SETGT_PUSH test (same as SETGE test with modification to the vtx shader). Also correcting description in the SETGE test

Change 109569 on 2003/07/07 by ashishs@fl_ashishs_r400_win2

updated tests as per new function put in by kevin ryan in primlib repacing Write_To_Memory function

Change 109553 on 2003/07/07 by ashishs@fl_ashishs_r400_win

shrinking more images for faster hardware run

Change 109549 on 2003/07/07 by ashishs@fl_ashishs_r400_win

shrinking test sizes to run faster in hardware

Change 109513 on 2003/07/07 by cbrennan@cbrennan_r400_win_marlboro

Fix 2d random cases from erroring out.
Enable tiled 2d unit test.

Change 109497 on 2003/07/07 by mkelly@fl_mkelly_r400_win_laptop

Update PrimLib function call to make new VB each packet.

Change 109462 on 2003/07/07 by cbrennan@cbrennan_r400_linux_marlboro

Turn off dumpPPMs

Change 109388 on 2003/07/05 by ashishs@fl_ashishs_r400_win

minor correction

Change 109387 on 2003/07/05 by ashishs@fl_ashishs_r400_win

updated test description

Change 109200 on 2003/07/03 by ashishs@fl_ashishs_r400_win

using PRED_SETE_PUSH description for PRED_SETNE_PUSH instruction

Change 109198 on 2003/07/03 by ashishs@fl_ashishs_r400_win

added much needed description to the test...

Change 109175 on 2003/07/03 by ashishs@fl_ashishs_r400_win

updated test_list and trackers

Change 109152 on 2003/07/03 by ashishs@fl_ashishs_r400_win

added tests

Change 109134 on 2003/07/03 by georgev@devel_georgev_r400_lin2_marlboro_tott

More support for larger textures.

Change 109125 on 2003/07/03 by ashishs@fl_ashishs_r400_win

These 2 tests changed after change #98332. I was accessing uninitialised color array value which started to give error color values when the service pack for VC was installed in dk_win folder. Hence corrected the color array to correctly have all color values and also kind of rearranged the colors so that in future could better identify if problem occurs.

Change 109105 on 2003/07/03 by cbrennan@cbrennan_r400_emu

Enable mip packing in ferret.
Enable mip packing in directed tests.
Mip packing bug fixes in HW.

Change 109063 on 2003/07/03 by georgev@devel_georgev_r400_lin2_marlboro_tott

Fixed texture offset.

Change 109062 on 2003/07/03 by georgev@devel_georgev_r400_lin2_marlboro_tott

Fixed texture offset.

Change 109016 on 2003/07/03 by cbrennan@cbrennan_r400_linux_marlboro

Fix testcase name mismatch.

Change 108972 on 2003/07/02 by omesh@omesh_r400_linux_marlboro_code_cover

Modified all tests to include full device address into all base address register programming. Did a simple compile for all

these tests, but haven't verified run time results.... Should be the same, as I have followed all primlib and register
guidelines.

Change 108919 on 2003/07/02 by ashishs@fl_ashishs_r400_win

Using the same test as in the clipper for line texture problem and disable clipping for SU

Change 108916 on 2003/07/02 by ashishs@fl_ashishs_r400_win

test samne as line_barycentric_clip_perspective_actualClip_01 but with 2 ucp clipping planes clipping off portions at desired locations

Change 108895 on 2003/07/02 by ashishs@fl_ashishs_r400_win

added some variables related to synching top level devel dir files and also to sync the dk_win dir. Also added the option to make clean and clobber

Change 108859 on 2003/07/02 by donaldl@donaldl_crayola_linux_orl

Added wait_gfx_idle after plgx init. Backed out previous change.

Change 108793 on 2003/07/01 by ashishs@fl_ashishs_r400_win2

changed the shader to go back to the failed status

Change 108788 on 2003/07/01 by georgev@devel_georgev_r400_lin2_marlboro_tott

Added new tests.

Change 108782 on 2003/07/01 by csampayo@fl_csampayo2_r400

Updated SU test for TP accuracy. Added ME_INT packet to enable RTS in VGT test

Change 108775 on 2003/07/01 by ashishs@fl_ashishs_r400_win

created test case according to Carlos's spec to determine problem in Clipper

Change 108758 on 2003/07/01 by hartogs@fl_hartogs2

Changed vertex reuse depth to a legal value.

Change 108755 on 2003/07/01 by omesh@omesh_r400_linux_marlboro_code_cover

Fixed a bug related to Framebuffer start address.

Change 108747 on 2003/07/01 by llefebvr@llefebvr_r400_emu_montreal

Bad GPR allocation in the shader was not allocating enough GPRS...

Change 108714 on 2003/07/01 by ashishs@fl_ashishs_r400_win2

changed pix shader to offset recent change in emulator and also run correctly on hardware

Change 108707 on 2003/07/01 by omesh@omesh_r400_linux_marlboro_code_cover

Allowed Framebuffer Base address to be relocatable by adding in the base
into the complete device address used by Color Buffer registers.
Verified that the framebuffer still dumps correctly. Will do the other
tests in batches.

Change 108695 on 2003/07/01 by cbrennan@cbrennan_r400_linux_marlboro

Removed size multiple of 4 restrictions from DXT.

Change 108688 on 2003/07/01 by cbrennan@cbrennan_r400_win_marlboro

Turned on more formats, gamma, and added testcase tri128_tex4_fb32x32

Change 108687 on 2003/07/01 by omesh@omesh_r400_linux_marlboro_only_devel

Added the missing SMASK enable programming. In the test, other stencil
related programming depended on this bit, so they should all work now.
Haven't tested the fix though. Relying on Paul Vella to try it out.

Change 108646 on 2003/06/30 by hartogs@fl_hartogs2

Reversing my previous check-in. Accidentally overwrote.

Change 108593 on 2003/06/30 by jayw@jayw_r400_linux_marlboro

Adding 16 bit pixel mask through DB. NOTE: db_stdrfsdks2p8x136cm1sw0 is now db_stdrfsdks2p8x152cm1sw0

Change 108576 on 2003/06/30 by jhoule@jhoule_r400_linux_marlboro

Now stripping executables...

Change 108559 on 2003/06/30 by omesh@omesh_r400_linux_marlboro_only_devel

Making surfaces compliant with Primlib's requirement for multiple of
tile sized surfaces to handle assertions. Haven't tested this yet but
need to recreate new Perforce client to move to new server.

Change 108540 on 2003/06/30 by csampayo@fl_csampayo2_r400

Added DISABLE_PERSPECTIVE and DEBUG controls and did some cleanup.

Change 108508 on 2003/06/30 by ashishs@fl_ashishs_r400_win

limiting num of vertcies to 128 to speed up the test

Change 108425 on 2003/06/27 by georgev@devel_georgev_r400_lin2_marlboro_tott

Added new tests.

Change 108423 on 2003/06/27 by ashishs@fl_ashishs_r400_win

initial checkin for PRED_SETGE_PUSH test

Change 108313 on 2003/06/27 by mkelly@fl_mkelly_r400_win_laptop

Temporarily remove r400sc_pm4_mp_ib_03 until invalidate VB bit is available
in VC.

Change 108276 on 2003/06/27 by tien@tien_r400_devel_marlboro

Fixed runme script yet again :-)
More formatter fixes

Change 108258 on 2003/06/27 by mkelly@fl_mkelly_r400_win_laptop

Create and use new VB on draw_initiator

Change 108257 on 2003/06/27 by mkelly@fl_mkelly_r400_win_laptop

Create and use new vertex buffer on each draw_initiator...

Change 108248 on 2003/06/27 by mkelly@fl_mkelly_r400_win_laptop

Fix case 11, add comments...

Change 108211 on 2003/06/26 by csampayo@fl_csampayo2_r400

Remove flushes and unnecessary waits between packets

Change 108197 on 2003/06/26 by ashishs@fl_ashishs_r400_win

Changing the vertex shader of the pres_sete_push to use the same data for comparisons to yield similar results...

Change 108181 on 2003/06/26 by tien@tien_r400_devel_marlboro

Formatter fixes

Change 108171 on 2003/06/26 by csampayo@fl_csampayo2_r400

Update to write vertex buffers to unique addresses

Change 108163 on 2003/06/26 by rmanapat@rmanapat_r400_sun_marlboro

Changed Clamp Policy for these tests except for the cubic tests which
already had the clamp_clamptolast policy set on the x and y coords

Change 108156 on 2003/06/26 by ashishs@fl_ashishs_r400_win

testing pred_sete_push instruction on the predicate value as well as the result in the output of the comparison

Change 108154 on 2003/06/26 by hartogs@fl_hartogs2

Hopefully fixed VGT alloc/dealloc for multi-SIMD vector sets
  Added simd_id fields to vgt_sq interface and vgt_ccgen interface
  Put pipedisable comments into several dump files.
  Put an "Assert(0)" into the sq_block_model.cpp to prevent access violation.

Change 108122 on 2003/06/26 by mkelly@fl_mkelly_r400_win_laptop

Add fmt_10_11_11 case...

Change 108090 on 2003/06/26 by mkelly@fl_mkelly_r400_win_laptop

fmt_10_11_11 on position, although all 3 verts are zero, TP returns non-zero data...

Change 108082 on 2003/06/26 by csampayo@fl_csampayo2_r400

Update to use only the 3 valid indices from VGT

Change 108080 on 2003/06/26 by tien@tien_r400_devel_marlboro

Fixed mini-regress script for release_* area usage
Fixed large unnuomed coord handling (for a cp_e2* test) .. for real this time I think)
Misc sp_tp_formatter fixes

Change 108075 on 2003/06/26 by mkelly@fl_mkelly_r400_win_laptop

Get it right this time

Change 108074 on 2003/06/26 by mkelly@fl_mkelly_r400_win_laptop

Changed name to keep consistent with other tests in this directory...

Change 108035 on 2003/06/26 by ashishs@fl_ashishs_r400_win

updated description in the shaders

Change 108010 on 2003/06/26 by mkelly@fl_mkelly_r400_win_laptop

Debug

Change 107968 on 2003/06/25 by ashishs@fl_ashishs_r400_win

entering COREECT description for CNDGT

Change 107965 on 2003/06/25 by ashishs@fl_ashishs_r400_win

entering the CORRECT DESCRIPTION for the test

Change 107961 on 2003/06/25 by ashishs@fl_ashishs_r400_win

Added CORRECT description for the test

Change 107949 on 2003/06/25 by vbhatia@vbhatia_r400_linux_marlboro

Example usage stimulus file for standalone vc_formatter testbench

Change 107928 on 2003/06/25 by ashishs@fl_ashishs_r400_win

editing constant data in r400sq_cnde_01 to create the test which has the same test logic to compare.

Change 107890 on 2003/06/25 by ashishs@fl_ashishs_r400_win

initial checkin for the CNDGE test. same as CNDE test, but changing the constant data so that the same logic of CNDE test could be used for the CNDGE test as well.

Change 107867 on 2003/06/25 by tien@tien_r400_devel_marlboro

Added back date to output

Change 107859 on 2003/06/25 by ashishs@fl_ashishs_r400_win

editing r400sq_const_index_03 for coverage to create const_index_04 with diff being in vtx_shader which uses MAX instead of MOV to putput color

Change 107852 on 2003/06/25 by ashishs@fl_ashishs_r400_win

wasnt multiplying color with the result of comparisons (only poistion)

Change 107790 on 2003/06/25 by cbrennan@cbrennan_r400_linux_marlboro

Added 3d til_fmt11-12_l back in.

Change 107776 on 2003/06/25 by tien@tien_r400_devel_marlboro

Updated check method to work for rel area too

Change 107731 on 2003/06/24 by ashishs@fl_ashishs_r400_win

updating test_list

Change 107730 on 2003/06/24 by ashishs@fl_ashishs_r400_win

16 textures and no color displayed on POLYGON with clipping disabled/enabled and toggling between TRI_FILL and LINE_FILL and also permuting PA_SU_SC_MODE_CNTL_persp_corr_dis with primitive setup in perspective view.

Change 107705 on 2003/06/24 by ashishs@fl_ashishs_r400_win

combo test having 1 color with 4 cases viz
tri_fill, pers_corr_dis(false)
tri_fill, pers_corr_dis(true)
line_fill, pers_corr_dis(false)
line_fill, pers_corr_dis(true)

same except clipping setup completely removed from
r400cl_polymode_persp_1color_combo_01 test.

Change 107703 on 2003/06/24 by ashishs@fl_ashishs_r400_win

combo test having 1 color with 4 cases viz
tri_fill, pers_corr_dis(false)
tri_fill, pers_corr_dis(true)
line_fill, pers_corr_dis(false)
line_fill, pers_corr_dis(true)

Change 107689 on 2003/06/24 by mdoggett@mdoggett_r400_linux_local

Forgot to update the stdout for sector mask 2 change.

Change 107688 on 2003/06/24 by mdoggett@mdoggett_r400_linux_local

Changed sector mask 2 by swapping right half of cacheline 8x2 to slice filter performance by giving the slice filter a 8x4 instead of the old 8x2 of contiguous pixels that an individual bilinear quad request can hit.

Change 107671 on 2003/06/24 by mkelly@fl_mkelly_r400_win_laptop

Add 1 case for FMT_16_16_FLOAT

Change 107668 on 2003/06/24 by ashishs@fl_ashishs_r400_win

initial checkin for CNDE test. The test checks 500 xyzw data on all components for cnde instruction with with switching between SRC B and SRC C registers as required by the test constarints.

Change 107665 on 2003/06/24 by tien@tien_r400_devel_marlboro

Added >1.0 mag clamp in formatter
A few bug fixes with normalizer mux
Added some features to tp_formatter regression script
Updated tp4_tc mini regression script

Change 107648 on 2003/06/24 by georgev@devel_georgev_r400_lin2_marlboro_tott

Removed tiling from 1D texture tests.

Change 107612 on 2003/06/24 by georgev@devel_georgev_r400_lin2_marlboro_tott

Removed shrinkage variable, to make tests larger.

Change 107611 on 2003/06/24 by georgev@devel_georgev_r400_lin2_marlboro_tott

Removed shrinking variable to bring tests up to full size.

Change 107550 on 2003/06/23 by ashishs@fl_ashishs_r400_win

testing polymode line_fill with textures

Change 107548 on 2003/06/23 by ashishs@fl_ashishs_r400_win

to test polymode line_fill with textures with clipping enabled for primitives...

Change 107498 on 2003/06/23 by omesh@omesh_r400_linux_marlboro_only_devel

Added several hundred HiZ/HiStencil test combos.
Also changed some testcase names from *_zfail_* to *_znever_* and
*_zpass_* to *_zalways_*

Change 107458 on 2003/06/23 by jayw@jayw_r400_linux_marlboro

right queues

Change 107432 on 2003/06/23 by rmanapat@rmanapat_r400_sun_marlboro

Temp changes to these tests that allow only even sized texture maps

Change 107424 on 2003/06/23 by jayw@jayw_r400_linux_marlboro

pre move

Change 107423 on 2003/06/23 by ashishs@fl_ashishs_r400_win

correcting test and making it eq to setne_01

Change 107419 on 2003/06/23 by ashishs@fl_ashishs_r400_win

updating trackers and test_list

Change 107280 on 2003/06/20 by vbhatia@vbhatia_r400_linux_marlboro

changes for new tp_formatter and vc_formatter standalone testbenches

Change 107273 on 2003/06/20 by ashishs@fl_ashishs_r400_win

forgot to remove some debug statements

Change 107272 on 2003/06/20 by ashishs@fl_ashishs_r400_win

updated to change as per Carlos's requirements and verifying the image with r300 which doesn't match currently

Change 107267 on 2003/06/20 by tien@tien_r400_devel_marlboro

Added run privs

Change 107257 on 2003/06/20 by tien@tien_r400_devel_marlboro

Complewted first pass and vc part of formatter
Made regression script more thorough

Change 107247 on 2003/06/20 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 107195 on 2003/06/20 by llefebvr@llefebvr_r400_emu_montreal

Forgot to place back the right exporting register.

Change 107194 on 2003/06/20 by llefebvr@llefebvr_r400_emu_montreal

Fixing shader, it was doing ALU operations in a texture clause.

Change 107152 on 2003/06/20 by ashishs@fl_ashishs_r400_win

changing the test to ouput 500 points instead of triangle list. Also removing textures from shaders and tests since wasn't necessary...

Change 107142 on 2003/06/20 by mdoggett@mdoggett_r400_linux_local

Added fetch_valid_only to pixel shader.
Switched test back to 256x256 at 0,0

Change 107102 on 2003/06/19 by ashishs@fl_ashishs_r400_win

removing texture setup since wasnt necessary. Also adding point_list instead of triangle_list and diplaying >500 points thereby passing enough data to fill up the shader pipe...

Change 107101 on 2003/06/19 by csampayo@fl_csampayo2_r400

Updated for better visualizing clamped addrs reg values

Change 106982 on 2003/06/19 by csampayo@fl_csampayo_r400

Initial checkin

Change 106981 on 2003/06/19 by ashishs@fl_ashishs_r400_win

Adding SETNE test...

Change 106954 on 2003/06/19 by mkelly@fl_mkelly_r400_win_laptop

Update test, but still doesn't get pixels in the RB due to many
recent changes...

Change 106944 on 2003/06/19 by mkelly@fl_mkelly_r400_win_laptop

Add test r400vgt_hos_pm4_01 to regression

Change 106943 on 2003/06/19 by mkelly@fl_mkelly_r400_win_laptop

Convert hos test from auto reg unions to shadow register useage.
Simplified includes further in vc data format 01.

Change 106897 on 2003/06/18 by tien@tien_r400_devel_marlboro

Added mip_packed output on tpc
More fixed to formatter

Change 106894 on 2003/06/18 by ashishs@fl_ashishs_r400_win

forgot to remove the unused subroutine

Change 106893 on 2003/06/18 by ashishs@fl_ashishs_r400_win

updating the test. Now checks all the 512 constants on all the components, generating the result in a predetermined fashion to detect any errors if any...

Change 106748 on 2003/06/18 by mkelly@fl_mkelly_r400_win_laptop

Translate test to use shadow registers...

Change 106733 on 2003/06/18 by llefebvr@llefebvr_r400_emu_montreal

changing parameters slightly.

Change 106712 on 2003/06/18 by jhoule@jhoule_r400_linux_marlboro

Moving from hicolor to tp_hicolor subdirectory.

Change 106708 on 2003/06/18 by jhoule@jhoule_r400_linux_marlboro

Now strips binaries.

Change 106694 on 2003/06/18 by jhoule@jhoule_r400_linux_marlboro

Now stripping binaries

Change 106687 on 2003/06/18 by smoss@smoss_crayola_win

added to cover min max register

Change 106686 on 2003/06/18 by mkelly@fl_mkelly_r400_win_laptop

Minor comment update...Also note previous version, case buffers
were modified to have more unique vertex and texture data per case.

Change 106685 on 2003/06/18 by mkelly@fl_mkelly_r400_win_laptop

Cleaned up some build warnings by initializing buffers, turned off
tiling in dump image.

Change 106647 on 2003/06/17 by ashishs@fl_ashishs_r400_win

initial checkin for the sete test

Change 106620 on 2003/06/17 by mkelly@fl_mkelly_r400_win_laptop

First VC test

Change 106619 on 2003/06/17 by mkelly@fl_mkelly_r400_win_laptop

First release, testing 4 data formats

Change 106606 on 2003/06/17 by georgev@devel_georgev_r400_lin2_marlboro_tott

Fixed multicontext texture tests.

Change 106556 on 2003/06/17 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 106533 on 2003/06/17 by llefebvr@llefebvr_r400_emu_montreal

Changing name of test per Mike Kelly's request.

Change 106529 on 2003/06/17 by llefebvr@llefebvr_r400_emu_montreal

Cylindrical wrap precision test.

Change 106489 on 2003/06/17 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 106484 on 2003/06/17 by mkelly@fl_mkelly_r400_win_laptop

Add support for VC

Change 106480 on 2003/06/17 by mkelly@fl_mkelly_r400_win_laptop

Remove empty line...

Change 106415 on 2003/06/16 by ashishs@fl_ashishs_r400_win

updating the test to perform all combinations of scalar operations like precision_mul_01 and also updating the test description

Change 106413 on 2003/06/16 by ashishs@fl_ashishs_r400_win

testing precision of MUL instruction. Also testing all combinations of scalar as well as vector operation in the same test.

Change 106389 on 2003/06/16 by csampayo@fl_csampayo_r400

Updated test description

Change 106371 on 2003/06/16 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 106333 on 2003/06/16 by mkelly@fl_mkelly_r400_win_laptop

Re-enable 3 tests that broke with non-zero FB addresses, fixed now.

Change 106332 on 2003/06/16 by mkelly@fl_mkelly_r400_win_laptop

Updates to make tests work with any framebuffer address

Change 106294 on 2003/06/16 by ashishs@fl_ashishs_r400_win

updated

Change 106291 on 2003/06/16 by ashishs@fl_ashishs_r400_win

updated

Change 106288 on 2003/06/16 by mkelly@fl_mkelly_r400_win_laptop

Remove 3 tests that break with new frame buffer address

Change 106280 on 2003/06/16 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 106279 on 2003/06/16 by mkelly@fl_mkelly_r400_win_laptop

Change to work with new FB start, added RTS_WAIT_UNTIL

Change 106278 on 2003/06/16 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 106107 on 2003/06/13 by ashishs@fl_ashishs_r400_win

testing precision for the add instruction

Change 106103 on 2003/06/13 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint, not ready for test

Change 106093 on 2003/06/13 by llefebvr@llefebvr_r400_emu_montreal

Emulator now clamps the address register to the range -256...255.

Change 106033 on 2003/06/13 by smoss@smoss_crayola_linux_orl

unix output is $user, and compressed

Change 105938 on 2003/06/12 by ashishs@fl_ashishs_r400_win

testing Constant memory to output buffers data path. Needed to explicitly allocate position and parameter in shader. Also needed to reassign the address register after position allloc ?

Change 105923 on 2003/06/12 by smoss@smoss_crayola_linux_orl

changed x,y to 128

Change 105886 on 2003/06/12 by ashishs@fl_ashishs_r400_win

testing setgt instruction

Change 105859 on 2003/06/12 by lseiler@lseiler_r400_win_marlboro

Changes GL_XXX blendop names to BLEND_XXX

Change 105828 on 2003/06/12 by ashishs@fl_ashishs_r400_win

testing shader instructions

Change 105818 on 2003/06/12 by ashishs@fl_ashishs_r400_win

initialising c31 explicitly to 0,0,0,0 since the constant wasn't initialised and was being used in the pixel shader. The VFD uses c31 as color0 if it doesn't find any color set in the vertex data.

Change 105736 on 2003/06/12 by jhoule@jhoule_r400_win_lt

Changed FMT_32_AS_8_8_INTERLACED to use FMT_8_8 as faux_format instead of FMT_8.

Change 105699 on 2003/06/11 by csampayo@fl_csampayo_r400

Add compare file

Change 105635 on 2003/06/11 by ashishs@fl_ashishs_r400_win

removing any code which could be the cause of suspicion for failure for the test. Just to show that the test still fails even without the previous suspicious code in the test.

Change 105608 on 2003/06/11 by mkelly@fl_mkelly_r400_win_laptop

Minor comment update...

Change 105606 on 2003/06/11 by mkelly@fl_mkelly_r400_win_laptop

VGT HOS PM4 conversion

Change 105512 on 2003/06/11 by mkelly@fl_mkelly_r400_win_laptop

Add more sq dumps

Change 105506 on 2003/06/11 by mkelly@fl_mkelly_r400_win_laptop

Add sq_shaders.dmp

Change 105409 on 2003/06/10 by csampayo@fl_csampayo2_r400

Fixed problem with Export_Base_Address being erroneously changed after first pass

Change 105369 on 2003/06/10 by mkelly@fl_mkelly_r400_win_laptop

Setup to regress for VC

Change 105365 on 2003/06/10 by mdoggett@mdoggett_r400_linux_local

added scissor, doing 2x2 top left quad point sampled

Change 105363 on 2003/06/10 by mkelly@fl_mkelly_r400_win_laptop

Add UseVc = 0 to both registery files

Change 105325 on 2003/06/10 by jhoule@jhoule_r400_linux_marlboro

Adding first tp_ch_blend binary.
Contains border bits as well as xor mask.

Updated Makefile to allow p4_releases.

Change 105318 on 2003/06/10 by jhoule@jhoule_r400_win_lt

Added border_color and xor mask. Removed data_format field, which is deprecated by the previous.
Updated example.in file to account for new columns.

Change 105074 on 2003/06/09 by csampayo@fl_csampayo_r400

Initialize directory

Change 105039 on 2003/06/09 by ashishs@fl_ashishs_r400_win

updated

Change 105030 on 2003/06/09 by tien@tien_r400_devel_marlboro

CHanged TP_SQ_fetch_stall to TP_SQ_dec

Change 105019 on 2003/06/09 by ashishs@fl_ashishs_r400_win

updated description in test

Change 105015 on 2003/06/09 by ashishs@fl_ashishs_r400_win

same as horz_01 but with texture offsets of 0.4 and 0.6 respectively

Change 104999 on 2003/06/09 by ashishs@fl_ashishs_r400_win

same as vert_01 but with texture offsets of 0.4 and 0.6 respectively

Change 104965 on 2003/06/09 by ashishs@fl_ashishs_r400_win

testing texture wrap over T with 64 cases each with a different level of wrapping from 1 to 64

Change 104944 on 2003/06/09 by llefebvr@llefebvr_r400_emu_montreal

Modified the emulator to generate the same number of passes than the HW on a waterfall instruction. Only the number of passes is correct at this point not the write mask.

Also corrected 2 MOVA tests where the address register wasn't loaded correctly before use.

Change 104937 on 2003/06/09 by mkelly@fl_mkelly_r400_win_laptop

Add r400sc_pm4_mp_ib_03

Change 104936 on 2003/06/09 by mkelly@fl_mkelly_r400_win_laptop

Add a couple of VC dumps to the regression maintenance

Change 104736 on 2003/06/06 by ashishs@fl_ashishs_r400_win

updated the tracker and test_list for the newly added 4 tests

Change 104729 on 2003/06/06 by csampayo@fl_csampayo2_r400

Updated for FB start move

Change 104719 on 2003/06/06 by ashishs@fl_ashishs_r400_win

testing texture wrap over S with 64 cases each with a different level of wrapping from 1 to 64

Change 104707 on 2003/06/06 by jhoule@jhoule_r400_linux_marlboro

Changed p4_release dependencies to strip the executable.

Change 104689 on 2003/06/06 by mkelly@fl_mkelly_r400_win_laptop

Update to make it easier to see memory management and work
with adjustable frame buffer start.

Change 104595 on 2003/06/06 by csampayo@fl_csampayo_r400

Update address for vtx shader load

Change 104594 on 2003/06/06 by georgev@devel_georgev_r400_lin2_marlboro_tott

Added new tests for TP testing.

Change 104449 on 2003/06/05 by ashishs@fl_ashishs_r400_win

This test is intended to validate the shader constant memory using up 256 constants in the vertex shader program

Change 104336 on 2003/06/05 by csampayo@fl_csampayo2_r400

Updates for new FB start

Change 104319 on 2003/06/05 by mkelly@fl_mkelly_r400_win_laptop

Emulator rbbm.dmp coverage tool. Useful info. to determine test coverage
on register bits. A bit is considered covered if it changes state from
0 to 1 or 1 to 0.

Simple steps to follow:
1. regress_r400
2. perl build_regspec.pl -b VGT
    3. regcover -r <root path of regress_r400 test dirs> -tl
4. results can be found in all_regcover.txt, regcover_warnings.txt

Change 104315 on 2003/06/05 by llefebvr@llefebvr_r400_emu_montreal

Nasty GPR allocation GPR error in the test that was causing the test to break on the HW.

Change 104312 on 2003/06/05 by ashishs@fl_ashishs_r400_win

testing exp instruction. Has problems with texture wrapping which is being looked into more closely

Change 104285 on 2003/06/05 by ashishs@fl_ashishs_r400_win

changing some of the texture coords to show the difference between r300 and r400 for texture

Change 104279 on 2003/06/05 by jhoule@jhoule_r400_linux_marlboro

Fixed missing clamp for out-of-range result of the final subtract in the correction.
Also added Linux binary to parts_lib/sim/gfx/tp.

Change 104271 on 2003/06/05 by tien@tien_r400_devel_marlboro

Aniso fixes
Aniso step order change

Change 104256 on 2003/06/05 by smoss@smoss_crayola_linux_orl

added picasso.h copy, rm *.h

Change 104246 on 2003/06/05 by jhoule@jhoule_r400_linux_marlboro

Removed countdown when releasing binary

Change 104203 on 2003/06/05 by ashishs@fl_ashishs_r400_win

texture coords to 64 and updated texture

Change 104202 on 2003/06/05 by ashishs@fl_ashishs_r400_win

simple test to show that currently texture wrapping currently has problem

Change 104192 on 2003/06/05 by mkelly@fl_mkelly_r400_win_laptop

Update.

Change 104191 on 2003/06/05 by mkelly@fl_mkelly_r400_win_laptop

Don't count a test name with sq in the sq category if it also is
named r400sp

Change 104157 on 2003/06/04 by ashishs@fl_ashishs_r400_win

updated the test_list fo SQ

Change 104132 on 2003/06/04 by ashishs@fl_ashishs_r400_win

lots of mixed instructions. seems to have problem with texture(commented instructionsin the shader)

Change 104067 on 2003/06/04 by jhoule@jhoule_r400_win_lt

Dedicated testbench for tp_hicolor.
Supports integer, float, and rf8.

Change 104049 on 2003/06/04 by jhoule@jhoule_r400_win_lt

Reactivated NEAREST_CLAMP_POLICY support.

Change 104025 on 2003/06/04 by csampayo@fl_csampayo2_r400

Add FB start offset at correct place

Change 103999 on 2003/06/04 by llefebvr@llefebvr_r400_emu_montreal

Missing a constant load (509)

Change 103973 on 2003/06/04 by mdoggett@MA_MDOGGETT_LT

Changed to 256 x 256 directly mapped bilinear.

Change 103925 on 2003/06/03 by csampayo@fl_csampayo2_r400

Update for moving framebuffer

Change 103919 on 2003/06/03 by danh@danh_crayola1_linux_orl

Changed *_SHADER_FILE file names.

Change 103917 on 2003/06/03 by danh@danh_r400_win

Changed the *_SHADER_FILE strings so they have the correct file names.

Change 103901 on 2003/06/03 by ashishs@fl_ashishs_r400_win

updated the shader and the .cpp file the way it was supposed to be according to r300. The shader had some instructions commented out and was exporting the 4th texture directly as it recieved instead of doing the calculations but now is turned back on with the same calculations as in r300 and works perfectly.

the reason they were commneted out were that that time I had not developed a subroutine for LIT instruction which was present in r300 but not on r400

Change 103896 on 2003/06/03 by ashishs@fl_ashishs_r400_win

changing some instructions as they were supposed to be...

Change 103895 on 2003/06/03 by ashishs@fl_ashishs_r400_win

test converted from r300. Has lots of instructions mixed together

Change 103890 on 2003/06/03 by georgev@devel_georgev_r400_lin2_marlboro_tott

New textures for testing.

Change 103853 on 2003/06/03 by llefebvr@llefebvr_r400_linux_marlboro

Making screen size 64x64 instead of 512x512 so test runs faster.

Change 103825 on 2003/06/03 by georgev@devel_georgev_r400_lin2_marlboro_tott

Modified for automatic files.

Change 103782 on 2003/06/03 by ashishs@fl_ashishs_r400_win

adding comments and extending the shaders for 7th and 8th textures

Change 103779 on 2003/06/03 by ashishs@fl_ashishs_r400_win

loop instruction test with random loop constants and some of loop "i" registers being reused.

Change 103728 on 2003/06/02 by ashishs@fl_ashishs_r400_win

initial checkin for the test

Change 103725 on 2003/06/02 by ashishs@fl_ashishs_r400_win2

adding R400ZeroBufferStart to the dumps registry script

Change 103688 on 2003/06/02 by georgev@devel_georgev_r400_lin2_marlboro_tott

Added a few comments.

Change 103685 on 2003/06/02 by georgev@devel_georgev_r400_lin2_marlboro_tott

First revision so others could use it.

Change 103660 on 2003/06/02 by paulv@paulv_r400_linux_marlboro

Changed r400_regression queue to r400_regression_ncv.

Change 103627 on 2003/06/02 by omesh@omesh_r400_linux_marlboro

Added another 1152 Stencil Pass ZFail testcases for Backface Stencil
modes.

Change 103622 on 2003/06/02 by omesh@omesh_r400_linux_marlboro

Added another 1152 Stencil Pass ZFail testcases.

Change 103620 on 2003/06/02 by ashishs@fl_ashishs_r400_win

corrected the files. I had exported a level higher than I was supposed to export from the registry resulting in a lot of other my own PC related data being exported too. I have fixed that and now these dumps have just the data which they should have. Sorry for the inconvenience and anyone using this if breaks please let me know.

Change 103614 on 2003/06/02 by mkelly@fl_mkelly_r400_win_laptop

Fix DISP_BASE to be offset from framebuffer address. If we want these tests to
work with the frame_buffer start address, the rb base registers must be changed
to use frame buffer start address and not DISP_BASE. Currently, these tests work
with the registery key R400ZeroBufferStart = 1.

Change 103566 on 2003/06/02 by mkelly@fl_mkelly_r400_win_laptop

Fix syntax error...

Change 103434 on 2003/05/30 by ashishs@fl_ashishs_r400_win

updated the test_list

Change 103394 on 2003/05/30 by ashishs@fl_ashishs_r400_win

changing the loop increment to match the vap_pvs_iLoop_02 test in R300 instead of vap_pvs_loop_02

Change 103359 on 2003/05/30 by ashishs@fl_ashishs_r400_win

testing constant memory addressing with several instruction types and different number of operands

Change 103335 on 2003/05/30 by georgev@devel_georgev_r400_lin2_marlboro_tott

Skewed more texture.

Change 103332 on 2003/05/30 by mkelly@fl_mkelly_r400_win_laptop

Change hard coded frame buffer start to use method Get_Start()...

Change 103331 on 2003/05/30 by ashishs@fl_ashishs_r400_win

Testing MULADD instruction with multiple accesses to input and constant memories

Change 103303 on 2003/05/30 by llefebvr@llefebvr_r400_emu_montreal

Fixing ashish's test.

Change 103288 on 2003/05/30 by ashishs@fl_ashishs_r400_win

Testing constant memory store indexing and addr registers and exporting using constant memory indexed by address registers.

Currently the test has a problem which has been expained in the vtx shader of the test.

Change 103277 on 2003/05/30 by tien@tien_r400_devel_marlboro

Mipmapping fixes

Change 103186 on 2003/05/29 by csampayo@fl_csampayo2_r400

Remove commented instructions

Change 103173 on 2003/05/29 by ashishs@fl_ashishs_r400_win

Testing constant memory store indexing and addr registers using various different swizzles

Change 103116 on 2003/05/29 by mkelly@fl_mkelly_r400_win_laptop

Fix syntax error introduced by script which changed DISP_BASE

Change 103103 on 2003/05/29 by llefebvr@llefebvr_r400_emu_montreal

Fixing Carlos's bug

Change 103100 on 2003/05/29 by ashishs@fl_ashishs_r400_win

testing precision for exp instruction. Found a lot of differences between the Hardware accurate shader emulator calculations wrt to the C compiler generated exp values. Fot the same data set the Non Hardware acurate emulator (using C compiler functions) generates exactly equal results (could be used for verification since both the values viz shader as well as the expected values are generted using the same functions)

Change 103086 on 2003/05/29 by omesh@omesh_r400_linux_marlboro_only_devel

Added 24_8_FLOAT versions of the ZFunction tests (49 testcases)

Change 103060 on 2003/05/29 by csampayo@fl_csampayo_r400

Added max constant memory addrs reg indexing test. Updated test_list and test tracker accordingly.

Change 103039 on 2003/05/29 by mkelly@fl_mkelly_r400_win_laptop

Only plgx::wait_rt_idle before frame buffer dump, implement rts_wait_until

Change 102988 on 2003/05/28 by ashishs@fl_ashishs_r400_win

testing precision of log_ieee instruction to 1/2**21 Also found that there is a possible precision loss (same as sqrt instruction) between 0 to 1 for values less than 10^-2. Found that for some values the comparison is exact between precomputed and shader computed values and for some values there is a difference in even the last 3 bits of the mantissa value. Also testing in the same test(same as sqrt test) if the instruction executes correctly on all the channels and also if the result is replicated on all the 4 channels.

Change 102944 on 2003/05/28 by smoss@smoss_crayola_linux_orl

add sp sx

Change 102941 on 2003/05/28 by smoss@smoss_crayola_linux_orl

add *.h files

Change 102923 on 2003/05/28 by mkelly@fl_mkelly_r400_win_laptop

Implement RTS_WAIT_UNTIL, minimizing plgx::wait_rt_idle(s)

Change 102912 on 2003/05/28 by ashishs@fl_ashishs_r400_win

updated the description to have more information on the problem cases. Also updated the VTE Y scale to multiply with the very small constant (1e-038) so that all the points lie within the Y dim of image

Change 102897 on 2003/05/28 by jhoule@jhoule_r400_linux_marlboro

Fixed early-terminated help string.

Change 102895 on 2003/05/28 by ashishs@fl_ashishs_r400_win

updated the description to have more information on the problem cases. Also updated the VTE Y scale to multiply with the very small constant (1e-025) so that all the points lie within the Y dim of image

Change 102856 on 2003/05/28 by ashishs@fl_ashishs_r400_win

testing precision of recipsq instruction to 1/2**21 Also found that there is a possible precision loss (same as sqrt instruction) between 0 to 1 for values less than 10^-2 . Also testing in the same test(same as sqrt test) if the instruction executes correctly on all the channels and also if the result is replicated on all the 4 channels.

Change 102838 on 2003/05/28 by ashishs@fl_ashishs_r400_win

testing precision of sqrt instruction to 1/2^21 Also found that there is a possible precision loss for values between 0 and 1 (exclusive) having values less than 10^-2. Currently some of those points are outside the display area wrt the precision constant.

Change 102835 on 2003/05/28 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 102832 on 2003/05/28 by mkelly@fl_mkelly_r400_win_laptop

Update for debugging...

Change 102821 on 2003/05/28 by mkelly@fl_mkelly_r400_win_laptop

Changed test to use the rbbm WAIT_RTS_IDLE before each RTS initiator
and only use plgx::wait_rt_idle before the framebuffer dump.

Change 102777 on 2003/05/27 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 102768 on 2003/05/27 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 102722 on 2003/05/27 by mkelly@fl_mkelly_r400_win_laptop

Update display base to start at beginning of frame_buffer.Get_Start()

Change 102717 on 2003/05/27 by omesh@omesh_r400_linux_marlboro_only_devel

Disabled Color Blending. It seems to produce all samples for Alpha =
1.0. Will do some further checking to find out if some other testcases
look fine.

Change 102698 on 2003/05/27 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 102426 on 2003/05/23 by jhoule@jhoule_r400_linux_marlboro

Removed the countdown under p4_release

Change 102399 on 2003/05/23 by georgev@devel_georgev_r400_lin2_marlboro_tott

Additional tests for MIP mapping and aniso.

Change 102392 on 2003/05/23 by ashishs@fl_ashishs_r400_win

adding test

Change 102391 on 2003/05/23 by ashishs@fl_ashishs_r400_win

changing the dataset to have more cases. Also changing the constant to 2^-23

Change 102382 on 2003/05/23 by ashishs@fl_ashishs_r400_win

changing the precision constant to 2^-23

Change 102363 on 2003/05/23 by ashishs@fl_ashishs_r400_win

Testing precision of trunc instruction (found to be same as the C compiler generated trunc value)
The test uses 500 points in a packet to carry the reciprocal values and visually identify the problem
locations. Tests both the vector as well as scalar operation.
This test sends predetermined test data as well as random data through the z channel
of the vertex data, as well as the precomputed TRUNC values through the y channel of the
vtx data. Hex values of the floating point numbers are evalauated and sent to avoid other
rounding issues. The shader then gets the data on the z channel and TRUNCS it,
compares with the precomputed TRUNC value passed in the y channel of vtx data,
and plots the RELATIVE difference after multiplying with the PRECISION CONSTANT (set to 2**42) on the Y axis
of the image. Also to better identify problem points, a color coding scheme is given to the points
as follows. The point that have difference of "0" between precomputed and shader computed values
get BLUE color. When the same difference is between 0 and <= 1, the points get GREEN color.
When the difference is greater than 1 which identifies them as problem points, they get RED color.

TBD : Some number combinations donot work, I guess due to rounding issues. Need to find the exact problem.
For now those numbers are commented out.
According to me the NAN's and INF's will still be needed to be checked seperately.

Change 102274 on 2003/05/23 by ashishs@fl_ashishs_r400_win

removed from SX and putting it in SP, since by mistake thought them to be in SX. Also updating the test_list for both blocks

Change 102260 on 2003/05/23 by ashishs@fl_ashishs_r400_win

updated for the new sx precision tests

Change 102240 on 2003/05/23 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 102198 on 2003/05/22 by mdoggett@mdoggett_r400_linux_local

Changes for testing tc cache.

Change 102171 on 2003/05/22 by ashishs@fl_ashishs_r400_win

checking scalar as well as vector floor instruction in the same test

Change 102166 on 2003/05/22 by ashishs@fl_ashishs_r400_win

Testing precision of floor instruction
The test uses 500 points in a packet to carry the reciprocal values and visually identify the problem
locations. This test sends predetermined test data as well as random data through the z channel
of the vertex data, as well as the precomputed FLOOR values through the y channel of the
vtx data. Hex values of the floating point numbers are evalauated and sent to avoid other
rounding issues. The shader then gets the data on the z channel and FLOORS it,
compares with the precomputed FLOOR value passed in the y channel of vtx data,
and plots the RELATIVE difference after multiplying with the PRECISION CONSTANT (set to 2**42) on the Y axis
of the image. Also to better identify problem points, a color coding scheme is given to the points
as follows. The point that have difference of "0" between precomputed and shader computed values
get BLUE color. When the same difference is between 0 and <= 1, the points get GREEN color.
When the difference is greater than 1 which identifies them as problem points, they get RED color. TBD : According to me the NAN's and INF's will still be needed to be checked seperately.

Change 102151 on 2003/05/22 by ashishs@fl_ashishs_r400_win

changing to pix center as 0

Change 102146 on 2003/05/22 by mkelly@fl_mkelly_r400_win_laptop

Final, 4 textures on an RT rectangle...

Change 102118 on 2003/05/22 by ashishs@fl_ashishs_r400_win

Testing precision of recip_ieee instruction to 1.0f/2**23

The test uses 500 points in a packet to carry the reciprocal values and visually identify the problem locations.
This test sends predetermined test data as well as random data through the z channel
of the vertex data, and the precomputed reciprocals of those values through the y channel of the
vtx data. Hex values of the floating point numbers are evalauated and sent to avoid other
rounding issues. The shader then gets the data on the z channel and reciprocates it,
compares with the precomputed reciprocal value passed in the y channel of vtx data,
and plots the RELATIVE difference after multiplying with the PRECISION CONSTANT to get the difference
between 0 and 1 and thereby be able to plot on the Y axis of the image. Also to better identify
problem points, a color coding scheme is given to the points as follows.
The point that have difference of "0" between precomputed and shader computed values

get BLUE color. When the same difference is between 0 and <= 1, the points get GREEN color.
When the difference is greater than 1 which identifies them as problem points, they get RED color.
TBD : According to me the NAN's and INF's will still be needed to be checked seperately.

Change 102069 on 2003/05/22 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint..

Change 102048 on 2003/05/22 by llefebvr@llefebvre_laptop_r400_emu

fixing test

Change 102046 on 2003/05/22 by mkelly@fl_mkelly_r400_win_laptop

Changed to gouraud shading...

Change 102018 on 2003/05/21 by csampayo@fl_csampayo_r400

Updated test description, test_list and test tracker

Change 102000 on 2003/05/21 by tien@tien_r400_devel_marlboro

Cleaned up clamping logic

Change 101947 on 2003/05/21 by mkelly@fl_mkelly_r400_win_laptop

Break out pipe disable tests into separate tests...

Change 101931 on 2003/05/21 by georgev@devel_georgev_r400_lin2_marlboro_tott

Added new MipMap tests.

Change 101901 on 2003/05/21 by ashishs@fl_ashishs_r400_win

updated for 2 new tests. (also corrected problem with no new line at the end of test_list)

Change 101895 on 2003/05/21 by mkelly@fl_mkelly_r400_win_laptop

Fix frame buffer start, fix RB asserts on samples

Change 101890 on 2003/05/21 by mkelly@fl_mkelly_r400_win_laptop

Fix eo_rt in test for HW

Change 101865 on 2003/05/20 by csampayo@fl_csampayo_r400

SImple address register constant indexing tests

Change 101845 on 2003/05/20 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 101830 on 2003/05/20 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 101786 on 2003/05/20 by mkelly@fl_mkelly_r400_win_laptop

Break out simd cases into separate tests to shorten run time...

Change 101784 on 2003/05/20 by mkelly@fl_mkelly_r400_win_laptop

Break out cases for simd into separate tests...

Change 101699 on 2003/05/19 by csampayo@fl_csampayo_r400

Update for coissue (dual) instructions

Change 101675 on 2003/05/19 by ashishs@fl_ashishs_r400_win

adding scalar as well as vector operation to the shader for checking, and also updating the .cpp file since the c compiler rounded some of the values

Change 101653 on 2003/05/19 by ashishs@fl_ashishs_r400_win

performing scalar as well as vector operation to check both

Change 101648 on 2003/05/19 by ashishs@fl_ashishs_r400_win

test for the floor instruction...

Change 101646 on 2003/05/19 by mkelly@fl_mkelly_r400_win_laptop

Break out simd cases into separate tests...

Change 101630 on 2003/05/19 by ashishs@fl_ashishs_r400_win

updated for the new perspective tests added...

Change 101629 on 2003/05/19 by mkelly@fl_mkelly_r400_win_laptop

Break out cases for simd testing

Change 101622 on 2003/05/19 by ashishs@fl_ashishs_r400_win

test with points setup on 4 corners of user defined frustum with varying z values. horizontal as well as vertical clipping applied on all the points using ucp planes

Change 101619 on 2003/05/19 by mdoggett@mdoggett_r400_linux_local

Added command line option to print cacheline and sector at beginning of line. Command line option -clinesectorprint must be used to turn on printing.

Change 101616 on 2003/05/19 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 101609 on 2003/05/19 by mkelly@fl_mkelly_r400_win_laptop

Break out simd cases to individual tests...

Change 101598 on 2003/05/19 by mkelly@fl_mkelly_r400_win_laptop

Memory violation occurs at the end of the second packet

Change 101579 on 2003/05/19 by ashishs@fl_ashishs_r400_win

test with points in perspective view having different z values and being clipped by ucp (different ucp clipping modes and setting point size in vtx and su registers)

Change 101419 on 2003/05/16 by tien@tien_r400_devel_marlboro

Moved input flops out of sp_tp_formatter

Change 101351 on 2003/05/16 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint

Change 101331 on 2003/05/16 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 101327 on 2003/05/16 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 101311 on 2003/05/16 by jhoule@jhoule_r400_win_lt

Updated golden images, without randoms.

Change 101310 on 2003/05/16 by jhoule@jhoule_r400_win_lt

Removed randomness in Set_Default_Values()

Change 101259 on 2003/05/15 by ashishs@fl_ashishs_r400_win

updated for r400sq_trunc_01

Change 101258 on 2003/05/15 by ashishs@fl_ashishs_r400_win

test for trunc instruction created as directed by Carlos. It shows that it currently has a problem with scalar trunc operation where it fails on -ve integer values like -1.0, -2.0.... but doesnot fail on fractional -ve integers. A part of code missing on the scalar side for the trunc operation.

Change 101137 on 2003/05/15 by csampayo@fl_csampayo_r400

Updated test status for tests:
r400sq_auto_wrapping_memories_01
r400sq_vs_memory_wrap_01
Sorted test_list

Change 101121 on 2003/05/15 by csampayo@fl_csampayo2_r400

Reduce screen size and update description

Change 101092 on 2003/05/14 by csampayo@fl_csampayo2_r400

Updated to wrap vertex and pixel shaders

Change 101048 on 2003/05/14 by jhoule@jhoule_r400_linux_marlboro

Added strip target to reduce binary size, as well as p4_* targets for emu_tb release.

Change 101046 on 2003/05/14 by jhoule@jhoule_r400_linux_marlboro

Added binary testbench into parts_lib

Change 101022 on 2003/05/14 by jhoule@jhoule_r400_win_lt

Added address library to Makefile under Windows.

Cleaned up golden.lst file.
- Sorted tests
- Removed duplicates
- Removed r400tc_simple_texture since Mike uses it for personnal tests

Change 100975 on 2003/05/14 by ashishs@fl_ashishs_r400_win

updated the tracker and test_list

Change 100972 on 2003/05/14 by mkelly@fl_mkelly_r400_win_laptop

Fix for SIMD

Change 100963 on 2003/05/14 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 100956 on 2003/05/14 by vbhatia@vbhatia_r400_linux_marlboro

Adding skeleton tp_formatter with standalone testbench

Change 100944 on 2003/05/14 by ashishs@fl_ashishs_r400_win

testing sqrt function by using 500 points with square root values between 0 and 2 to plot the sqrt graph

Change 100932 on 2003/05/14 by mkelly@fl_mkelly_r400_win_laptop

Update eo_const_rt, ps_const.base and size

Change 100900 on 2003/05/14 by csampayo@fl_csampayo2_r400

New instruction store wrap test. Updated test_list accordingly.

Change 100890 on 2003/05/14 by mkelly@fl_mkelly_r400_win_laptop

Made test less weird :)

Change 100811 on 2003/05/13 by ashishs@fl_ashishs_r400_win

using z and w channels in the shader to do the square root. Need to verify if the w channel is selected by default...

Change 100797 on 2003/05/13 by ashishs@fl_ashishs_r400_win

cleaning up code...

Change 100781 on 2003/05/13 by ygiang@ygiang_r400_pv2_marlboro

fixed: sp tests emulation errors

Change 100779 on 2003/05/13 by ashishs@fl_ashishs_r400_win

correcting an error in the test

Change 100778 on 2003/05/13 by mkelly@fl_mkelly_r400_win_laptop

All kinds of RT constant indexing and clamping, using all 256 bool bits

Change 100775 on 2003/05/13 by jhoule@jhoule_r400_win_lt

Changed block name to tc (was perf).

Change 100760 on 2003/05/13 by ashishs@fl_ashishs_r400_win

another SQRT test, same as r400sq_sqrt_ieee_comp_02, but multiplying the vertex data with nan and inf data in x channel and then passing the inverse nan_multiplier in z channel to get the proper vertex data back.

Change 100719 on 2003/05/13 by jhoule@jhoule_r400_linux_marlboro

Fixed early-terminated help string.

Change 100715 on 2003/05/13 by ashishs@fl_ashishs_r400_win

another sqrt test. plotting the difference in precision between shader and c++ complier for square root instruction in the image.

Change 100702 on 2003/05/13 by csampayo@fl_csampayo2_r400

Adjust the SQ_PS_CONST register BASE and SIZE fields settings as per RT requirements

Change 100700 on 2003/05/13 by mkelly@fl_mkelly_r400_win_laptop

Finalize RT constant indexing positive and negative clamping variations

Change 100684 on 2003/05/13 by kevino@kevino_r400_linux_marlboro

Got rid of til fmt 11 and 12 testcases for the 3D tests. These are not valid 3D formats.

Change 100643 on 2003/05/13 by ashishs@fl_ashishs_r400_win

testing sqrt instruction

Change 100432 on 2003/05/12 by jhoule@jhoule_r400_linux_marlboro

Adding testbench binary for tp_addresser.
Updated Makefile's p4 targets.

Change 100401 on 2003/05/12 by jhoule@jhoule_r400_ma-jhoule-linux

Point sampling now sets incr_{x|y} to 0, just like hardware.
Added p4_release target to be used for putting the binary under Perforce.

Change 100397 on 2003/05/12 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 100273 on 2003/05/09 by csampayo@fl_csampayo2_r400

Initial check in

Change 100246 on 2003/05/09 by ashishs@fl_ashishs_r400_win

updated the excel tracker sheet and the test_list up-to-date

Change 100229 on 2003/05/09 by ashishs@fl_ashishs_r400_win

This test processes 112 packets of 4 vertices each. The pvs program uses the address register to calculate color0.
For each packet the test changes the parameter used for the address calculation randomly, for each of the four vertices.
Addressing of constant memory spans locations 0 thru 255

Change 100218 on 2003/05/09 by omesh@omesh_r400_linux_marlboro

Enabled ZMASK for multisampling rendering, as described in the RB
Register specs.

Change 100215 on 2003/05/09 by omesh@omesh_r400_linux_marlboro

Enabled ZMASK for multisample rendering, as described in the RB Register
specs.

Change 100214 on 2003/05/09 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 100209 on 2003/05/09 by omesh@omesh_r400_linux_marlboro

Enabled ZMASK in these tests, as Stencil Compression does require ZMASK
to be enabled.

Change 100173 on 2003/05/09 by mkelly@fl_mkelly_r400_win_laptop

Fix ps_const and eo_rt variables to work in HW.

Change 100143 on 2003/05/09 by ashishs@fl_ashishs_r400_win

This test processes 18 packets of 1 vertex each. The pvs program uses the address register to calculate color0. For each packet the test changes the parameter used for the address calculation for each vertex.

Change 100138 on 2003/05/09 by ashishs@fl_ashishs_r400_win

This test processes 6 packets of 2 vertices each. The pvs program uses the address register to calculate color0.For each packet the test changes the parameter used for the address calculation for each of the two vertices, permuting all possibilities where some of the parameters are equal to each other.

Change 100130 on 2003/05/09 by ashishs@fl_ashishs_r400_win

This test processes 10 packets of 3 vertices each. The pvs program uses the address register to calculate color0. For each packet the test changes the parameter used for the a0 calculation for each of the three vertices, permuting all possibilities where some of the parameters are equal to each other, while the remaider are also equal to each other

Change 100126 on 2003/05/09 by ashishs@fl_ashishs_r400_win

r400sq_addrs_reg_waterfall_03: This test processes 10 packets of 3 vertices each. The pvs program uses the addr register to calculate color0. For each packet the test changes the parameter used for the a0 calculation for each of the three vertices, permuting all possibilities where some of the parameters are equal to each other, while the remaider are not equal to each other
r400sq_addrs_reg_waterfall_02: cleaned up the test

Change 100120 on 2003/05/09 by ashishs@fl_ashishs_r400_win

This test processes 18 packets of 4 vertices each. The shader program uses the address register to calculate color0. For each packet the test changes the parameter used for the address calculation for each of the four vertices, permuting all possibilities where some of the parameters are equal to each other, while the remaider are also equal to each other.

Change 99995 on 2003/05/08 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 99994 on 2003/05/08 by ashishs@fl_ashishs_r400_win

somehow again there were some incorrect names...

Change 99993 on 2003/05/08 by ashishs@fl_ashishs_r400_win

somehow there was a wrong test name...

Change 99982 on 2003/05/08 by jhoule@jhoule_r400_win_lt

Changed opcode_enc from 2 hex to a single one (since it's 3b)

Change 99971 on 2003/05/08 by llefebvr@llefebvr_r400_emu_montreal

test had an error...

Change 99966 on 2003/05/08 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 99953 on 2003/05/08 by mkelly@fl_mkelly_r400_win_laptop

Zero out eo_rt and ps_const registers at test beginning..

Change 99948 on 2003/05/08 by omesh@omesh_r400_linux_marlboro

Added 1152 Backface versions of the same Multisample tests.

Change 99872 on 2003/05/08 by ashishs@fl_ashishs_r400_win

removing a small error with the script which made it to loop infinitely

Change 99807 on 2003/05/07 by jhoule@jhoule_r400_win_lt

Added tp_addresser testbench.

tp_types.h
- Added PackedCoord_32, intented to be shared by addresser as well as TP_TC interface

Change 99797 on 2003/05/07 by mdoggett@mdoggett_r400_linux_local

Removed setting frame buffer to 256M. In attempt to fix emulator assert on tptc testbench.
General clean up of file.

Change 99795 on 2003/05/07 by ashishs@fl_ashishs_r400_win

added variables for the AUTO_R400_REGRESS_getError script. The variables have to be present while executing that script or else not needed.

Change 99775 on 2003/05/07 by ashishs@fl_ashishs_r400_win

The script will now check the boundary values if at the end of the script if the test fails all over the range and then give the user appropriate error message. Changed some formatting also. Updated the description for the test in the header.
For more information on the script pls see the script header.

Change 99652 on 2003/05/07 by mkelly@fl_mkelly_r400_win_laptop

This should fix regress_e, I needed to update the pix shader to
use k0 instead of c0 since that is the way I set up the constant
store for RT.

Change 99646 on 2003/05/07 by mkelly@fl_mkelly_r400_win_laptop

I fixed the assert by programming eo_rt,ps_cons.size and base but
now image is not right, will continue debugging...

Change 99593 on 2003/05/07 by mkelly@fl_mkelly_r400_win_laptop

Nice Assert, reverted back to original test...

Change 99589 on 2003/05/07 by omesh@omesh_r400_linux_marlboro

Changed an earlier enumeration to match a revised emulator enumeration.
Verified that Primlib now supports the .Fill_Data function for
multisample depth correctly and verified that the dumped file looks ok.

Change 99566 on 2003/05/07 by ashishs@fl_ashishs_r400_win

sorry ...a bad error due to my mistake

Change 99564 on 2003/05/07 by ashishs@fl_ashishs_r400_win

removed some small errors again relating to email: the attachment of log_file with email

Change 99556 on 2003/05/07 by llefebvr@llefebvr_r400_emu_montreal

fixing the RT assertion.

Change 99551 on 2003/05/07 by ashishs@fl_ashishs_r400_win

removed some small errors...

Change 99548 on 2003/05/07 by ashishs@fl_ashishs_r400_win

logging actions to a log file as the script progresses...

Change 99547 on 2003/05/07 by mkelly@fl_mkelly_r400_win_laptop

Update comments...

Change 99518 on 2003/05/07 by mkelly@fl_mkelly_r400_win_laptop

Positive and Negative Constant Index clamping for RT...

Change 99435 on 2003/05/06 by georgev@devel_georgev_r400_lin2_marlboro_tott

Fix some of the missing tests.

Change 99385 on 2003/05/06 by mkelly@fl_mkelly_r400_win_laptop

Update comments, new tests checking RT Constant indexing...

Change 99375 on 2003/05/06 by ygiang@ygiang_r400_pv2_marlboro

fixed: VTXs out side of frame buffer problem for sp tests

Change 99344 on 2003/05/06 by mkelly@fl_mkelly_r400_win_laptop

Update comments on 06, finalized 07

Change 99319 on 2003/05/06 by ashishs@fl_ashishs_r400_win

can now accept command line parameter "To Group" to be used as group values while running the script
usage
perl AUTO_R400_REGRESS_getError [group1,group2....]

p.s. groups have been defined in the pa_regress/email_list

Change 99317 on 2003/05/06 by mdoggett@mdoggett_r400_linux_local

Fixed two layer formats to swizzle x,y in the 2nd layer.

Change 99286 on 2003/05/06 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 99281 on 2003/05/06 by smoss@smoss_crayola_linux_orl

fixed small error

Change 99257 on 2003/05/05 by ashishs@fl_ashishs_r400_win

also uncommenting the print to screen option

Change 99256 on 2003/05/05 by ashishs@fl_ashishs_r400_win

forgot to change the MIME directory permissions in the script for the email to work..

Change 99223 on 2003/05/05 by ashishs@fl_ashishs_r400_win

adding time related details to the output viz start time, end time, elapsed time for each time the script performs sync-make-regress . Also the script based on the values gives the sync numbers on which the test status has changed. (But I would prefer to look at the log of all the sync number runs in the email to be sure)

Change 99195 on 2003/05/05 by georgev@devel_georgev_r400_lin2_marlboro_tott

Added new anisotropic tests.

Change 99178 on 2003/05/05 by kevino@kevino_r400_linux_marlboro

Removed setting of max mip level to 5 or 6 to keep tests from going to the x1 mip level.

Change 99136 on 2003/05/05 by mkelly@fl_mkelly_r400_win_laptop

Change reg naming for pipe disable...

Change 99130 on 2003/05/05 by smoss@smoss_crayola_linux_orl

added scissor to remove rb asserts

Change 99117 on 2003/05/05 by mkelly@fl_mkelly_r400_win_laptop

Change Bad pipe reg naming

Change 99110 on 2003/05/05 by mkelly@fl_mkelly_r400_win_laptop

Change bad pipe reg naming / field naming...

Change 99101 on 2003/05/05 by mkelly@fl_mkelly_r400_win_laptop

Update nonRT shader names...

Change 99098 on 2003/05/05 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 99082 on 2003/05/05 by smoss@smoss_crayola_linux_orl

add *.h

Change 99068 on 2003/05/05 by mkelly@fl_mkelly_r400_win_laptop

SQ RT constants and flow control testing

Change 98922 on 2003/05/02 by ashishs@fl_ashishs_r400_win

adding few more template variables for the AUTO_R400_REGRESS_getError script

Change 98921 on 2003/05/02 by ashishs@fl_ashishs_r400_win

added to do list

Change 98918 on 2003/05/02 by ashishs@fl_ashishs_r400_win

checkin for initally changing the file permissions on some files to writable and then at the end of the script change the file permissions to read only.

Change 98917 on 2003/05/02 by csampayo@fl_csampayo2_r400

New channel mask tests

Change 98911 on 2003/05/02 by kryan@kryan_r400_win_marlboro_XP

chip/gfx/tc/ golden images

chip/gfx/tc/Makefile

Add basic gold files for standard test cases for each of the

tests in the gfx/tc branch. This is to aid the regression

of the PrimLib library to make sure 1d, 2d, and 3d textures

always work.

Ideally these golden images should be updated by people working

on the TC, but will at least be updated if PrimLib regression

finds a difference in them that is supposed to be there.

Also updated the tc/Makefile to add the tests so that they

can all be run with the command make emu.

These tests are NOT and should NOT be added to the Emulator regression.

Change 98908 on 2003/05/02 by jhoule@jhoule_r400_ma-jhoule-linux

Added 'copy' target to help binary releases.

Change 98874 on 2003/05/02 by ashishs@fl_ashishs_r400_win

Initial checkin for Script used for finding the error sync(sync on which the test fails). Depending on the range provided (pass..fail range), it will search for the sync at which the test started first failing. This is done using the binary search with the starting range as the PASS range and the ending range as the FAIL range. Currently, according to my estimates the script takes an less than an hour to find out the error sync on average, but I guess with more testing and heuristics this should be decreased.

Please also consider that you have to take in consideration of the range values while taking in account time taken for this script. The estimated time is less than 1 hour for a range of 1000 sync's(93000..94000 pass..fail range) currently.

Change 98859 on 2003/05/02 by mmang@mmang_crayola_linux_orl

Fix compile - bad previous merge

Change 98858 on 2003/05/02 by mkelly@fl_mkelly_r400_win_laptop

Remap pipe disable

Change 98852 on 2003/05/02 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 98850 on 2003/05/02 by mkelly@fl_mkelly_r400_win_laptop

Remap for new pipe disable registers...

Change 98830 on 2003/05/02 by mkelly@fl_mkelly_r400_win_laptop

Fix test bug where RT prim was going outside of surface...

Change 98829 on 2003/05/02 by omesh@omesh_r400_linux_marlboro

Removed multiple occurences of ZRange enabling.

Change 98824 on 2003/05/02 by omesh@omesh_r400_linux_marlboro

Enabled Z Range, in order to enable HiZ.

Change 98819 on 2003/05/02 by viviana@viviana_crayola_linux_orl

Changed the values written into PA_SC_AA_CONFIG.MSAA_NUM_SAMPLES to a legal value.

Change 98815 on 2003/05/02 by omesh@omesh_r400_linux_marlboro

Instead of disabling Z_ENABLE (this would not allow me access to the STENCILZFAIL operation), I enabled HiZ, so that both ZPass and ZFail are possible outcomes, depending on the specific testcase.

Change 98804 on 2003/05/02 by mkelly@fl_mkelly_r400_win_laptop

Change pipe disable register and fields...

Change 98803 on 2003/05/02 by csampayo@fl_csampayo2_r400

Update

Change 98800 on 2003/05/02 by csampayo@fl_csampayo2_r400

Position and color channel mask and predicate mask tests

Change 98798 on 2003/05/02 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 98791 on 2003/05/02 by jhoule@jhoule_r400_win_lt

Changed GRAD_EXP_ADJUST from 3b (instr) to 5b (const).

Change 98782 on 2003/05/02 by llefebvr@llefebvr_r400_emu_montreal

Fixing RT read constant test. Also fixed the eo_rt register to 8 bits (max aperture prevents us to go bigger anyways).

Change 98777 on 2003/05/02 by ashishs@fl_ashishs_r400_win

deleting and building test_list from blocks only if respective variables for regressions is turned ON. If all of the regression variables are off then by defualt the script uses the test_list in the pa_regress directory

Change 98773 on 2003/05/02 by mmang@mmang_crayola_linux_orl

1. Added constant address register valids to validate the address register data. The valid is set when address register is written. If valid is not set, sequencer will not waterfall those vertices or pixels. This disables waterfalling for predicated off writes and improperly initialized contant address registers.
2. Fixed bug in sqs_alu_instr_seq for phase 3 snooping of constant address registers bus. Previously, this snooping did not account for predication of those registers.
3. Fixed bug where ais_load_done_bits was not hooked up. This signal disables previous vector/scalar management which needs to be turned off during constant waterfalling. With bug, pvps logic went unknown which caused unknowns to eventually propagate in and out of the gprs.
4. Fixed bug where non-optimized offset was not being determined properly. non_opt_offset is determined by a priority encoder of p0_done, p1_done, p2_done, and p3_done.
5. With advent of constant address register valids, created waterfall_active_q to properly init and avoid re-initing of different pixel and vertex done bits.

Change 98758 on 2003/05/02 by kevino@kevino_r400_linux_marlboro

modified for mini regress

Change 98756 on 2003/05/02 by mkelly@fl_mkelly_r400_win_laptop

Change bad pipe register and field names...

Change 98753 on 2003/05/02 by kevino@kevino_r400_linux_marlboro

set max mip lvel so we never get to a mip level with a texture width/height of 1. The emu tp_tc has problems with this currently. Remove once this is fixed.

Change 98745 on 2003/05/02 by mkelly@fl_mkelly_r400_win_laptop

Fix bug in test where RT prim was written outside of Surface extents in RB.

Change 98673 on 2003/05/01 by georgev@devel_georgev_r400_lin2_marlboro_tott

Fix z buffer problem.

Change 98672 on 2003/05/01 by mdoggett@mdoggett_r400_linux_local

Added RF expand from 13 bit output from DXN to 16 bits.
1D linear for 4x16 expanding formats didn't do correct 1d linear ordering.
Fixed RF expand for 16 16 16 16 EXPAND. It was commented out for unknown reason.

Change 98651 on 2003/05/01 by llefebvr@llefebvr_r400_emu_montreal

minor update... not fixed...

Change 98622 on 2003/05/01 by mkelly@fl_mkelly_r400_win_laptop

Check clamping on RT loop indexing...

Change 98603 on 2003/05/01 by georgev@devel_georgev_r400_lin2_marlboro_tott

Changed for max loops macro.

Change 98602 on 2003/05/01 by georgev@devel_georgev_r400_lin2_marlboro_tott

Changed for new max_loops macro.

Change 98600 on 2003/05/01 by georgev@devel_georgev_r400_lin2_marlboro_tott

Changed for new max_loops variables.

Change 98594 on 2003/05/01 by georgev@devel_georgev_r400_lin2_marlboro_tott

Removed some mc_motion stuff.

Change 98591 on 2003/05/01 by omesh@omesh_r400_linux_marlboro

Adding 55 Alpha to Sample Mask tests. However, have to clarify the expected output for some testcases from Larry to confirm that the emulator is doing the right thing. H/W isn't ready yet, so I'm not adding these tests to the nightly regression.

Change 98558 on 2003/05/01 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 98549 on 2003/05/01 by scarter@scarter_emul_r400_linux_marlboro

Per George V.

Added:
#include <primlib/shader_program/control_flow/loop_id.h>

Changed:
CONTROL_DEFINITION_TABLE::MAX_LOOP_ID
to
LOOP_ID::MAX_ID

Change 98527 on 2003/05/01 by ashishs@fl_ashishs_r400_win

updated the regress_r400 script to always write to the email_contents file.

Change 98468 on 2003/05/01 by markf@markf_r400_lt_marlboro

Fixed tp_unsigned32_01.cpp to dump 1D surfaces correctly.

Change 98454 on 2003/05/01 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 98438 on 2003/04/30 by ashishs@fl_ashishs_r400_win

corrected a minor error with the script and also updated the variables file to add the new variable compare_list which will be used to copy the compare list from the respective directory

Change 98437 on 2003/04/30 by ashishs@fl_ashishs_r400_win

adding code to use compare_list from a regression directory based on the value of the variable set in auto regress. If the variable not set then uses the compare_list from the current directory

Change 98354 on 2003/04/30 by omesh@omesh_r400_linux_marlboro

Fixed this particular emulator assertion by disabling Color and Alpha blending for non blendable color formats. Verified that the particular seeded random testcase no longer asserts.

Change 98345 on 2003/04/30 by ashishs@fl_ashishs_r400_win

forgot to put a comment

Change 98338 on 2003/04/30 by georgev@devel_georgev_r400_lin2_marlboro

Added to help coverage.

Change 98337 on 2003/04/30 by georgev@devel_georgev_r400_lin2_marlboro

Added some cases to cover neg infinity and max negative numbers.

Change 98303 on 2003/04/30 by ashishs@fl_ashishs_r400_win

In the variables file if the sync_number is set to "0" viz.
set sync_number = "0"
then the script wont perform sync and make but will directly jump to the regressions

Change 98281 on 2003/04/30 by llefebvr@llefebvr_r400_emu_montreal

Using 4 loops but only setting up one... I have corrected the shader to use only loop 0.

Change 98273 on 2003/04/30 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 98264 on 2003/04/30 by mkelly@fl_mkelly_r400_win_laptop

Check point...

Change 98258 on 2003/04/30 by ashishs@fl_ashishs_r400_win

added variables for SP and SX regressions

Change 98255 on 2003/04/30 by ashishs@fl_ashishs_r400_win

added support for SP and SX regressions

Change 98249 on 2003/04/30 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint....

Change 98247 on 2003/04/30 by mkelly@fl_mkelly_r400_win_laptop

Add support for SP and SX regressions...

Change 98216 on 2003/04/29 by smoss@smoss_crayola_linux_orl

update

Change 98202 on 2003/04/29 by csampayo@fl_csampayo2_r400

Initial check-in

Change 98201 on 2003/04/29 by csampayo@fl_csampayo2_r400

Fine tuned this test to simplify and add more failure detection at output.

Change 98176 on 2003/04/29 by omesh@omesh_r400_linux_marlboro

Added the 1152 Stencil Multisample tests. However, the initialization for the Stencil per sample is not correct:
1) Primlib's Fill_Data routine is not populating the entire multisampled Depth buffer (Will file a Bugzilla for this soon. Kevin already knows about this).
2) Haven't completely verified if Larry's Address routine is corrrectly addressing each sample of the Depth Multisample buffer.
Will not add these testcases to nightly regression yet because of the above reasons. Also, these 1152 testcases only represent 1/32nd of the total Stencil Multisample directed testcase possibilities. If a scheme of cycling testcase groups comes up soon that would be nice, else many test conditions may go untested.

Change 98175 on 2003/04/29 by markf@markf_r400_lt_marlboro

Temp change to tc_simple_1d.cpp and tc_simple_2d.cpp randoms to only use maps w/ even dimensions.

Change 98162 on 2003/04/29 by mkelly@fl_mkelly_r400_win_laptop

Maximum pixel shader nested, control flow subroutines in RTS, with non-RTS in front and back containing simple pixel and vertex shaders.

Updated SQ doc with RTS tests needed.

Change 98121 on 2003/04/29 by ygiang@ygiang_r400_linux_marlboro

fixed: test is using indirect buffer now

Change 98097 on 2003/04/29 by markf@markf_r400_lt_marlboro

Added 24_8 to tc_simple_mip_2d.cpp

Change 98087 on 2003/04/29 by mkelly@fl_mkelly_r400_win_laptop

Copied in Makefile from VGT for consistency and to fix a build problem.

Change 98051 on 2003/04/29 by markf@markf_r400_lt_marlboro

Adding trilinear performance test

Change 98004 on 2003/04/28 by csampayo@fl_csampayo_r400

Initial check-in

Change 97980 on 2003/04/28 by mdoggett@mdoggett_r400_linux_local

Fixed bug in Source Address for 32_AS_8_INTERLACE 2D tiled.

Change 97908 on 2003/04/28 by omesh@omesh_r400_linux_marlboro

Added 30 multisample testcases that use color blending (SRC+DEST).

Change 97883 on 2003/04/28 by markf@markf_r400_lt_marlboro

Removed some TC performance tests, added some others.

Change 97711 on 2003/04/25 by dougd@dougd_r400_linux_marlboro

changed the shader programs for the *_max_aluconst test cases in order to cause the alu constant store to be read

Change 97640 on 2003/04/25 by markf@markf_r400_lt_marlboro

Added more full set of formats to 3D map tc tests

Change 97602 on 2003/04/25 by mdoggett@mdoggett_r400_linux_local

Added DXT2345_as_4x16.
Fixed (hopefully) 1bpp as 2d tiled.

Change 97596 on 2003/04/25 by ashishs@fl_ashishs_r400_win

deleting the registry export since can be done through all dumps off and on

Change 97595 on 2003/04/25 by ashishs@fl_ashishs_r400_win

correcting the TP_NoMipPointRound value to the correct key

Change 97592 on 2003/04/25 by ashishs@fl_ashishs_r400_win

inputting the registry key for TP_NoMipPointRound

Change 97546 on 2003/04/24 by omesh@omesh_r400_linux_marlboro

Fixed extent of x, y fan rendering to screen_width_x - 1 and screen_height_y - 1 in the random number generator.
Verified that this fixed atleast 1 of the asserting testcases in the last nightly regression and it will probably fix all 31 others too.

Change 97545 on 2003/04/24 by mdoggett@MA_MDOGGETT_LT

Added DXT1_AS_4x16 2D Linear.
Added DXT2_3/4_5_AS_4x16 - not tested.

Change 97524 on 2003/04/24 by ashishs@fl_ashishs_r400_win

updated

Change 97517 on 2003/04/24 by ashishs@fl_ashishs_r400_win

another flt2fix test

Change 97504 on 2003/04/24 by ashishs@fl_ashishs_r400_win

cleaned up code

Change 97500 on 2003/04/24 by ashishs@fl_ashishs_r400_win

Float to fix conversion for address register loads

Change 97483 on 2003/04/24 by mdoggett@mdoggett_r400_linux_local

Added DXT1_AS_4x16 2D tiled.

Change 97409 on 2003/04/24 by ashishs@fl_ashishs_r400_win

Needed to put scissor settings for the tests. The tests caused assertion due to

Change 96878 by lseiler@lseiler_r400_win_marlboro on 2003/04/22 10:34:50
Asserts if the tile (X,Y) address is beyond (pitch, height).

Change 97246 on 2003/04/23 by ashishs@fl_ashishs_r400_win

adding laurent to the email groups

Change 97228 on 2003/04/23 by vgoel@fl_vgoel2

changed test for loading texture image early in the program and changed texture register to T3 for fetching

Change 97226 on 2003/04/23 by omesh@omesh_r400_linux_marlboro

Added 144 Multisample Z tests. Next on the list is Multisample Stencil tests. These will be run in
tonight's regression. Looked at the emulator output of 1 or 2 testcases, but haven't completely
visually verified most testcases.

Change 97139 on 2003/04/23 by jayw@jayw_r400_linux_marlboro

updates

Change 97113 on 2003/04/23 by jayw@jayw_r400_linux_marlboro

AB removal and LEDA fixes, fix for 3 and 6 sample MSAA.

Change 97072 on 2003/04/23 by mkelly@fl_mkelly_r400_win_laptop

Access full range of GPRs, 32 Booleans, Control Flow, 256 RT Constants
4 levels of nested subroutines, Looping, Maximum pixel shader instruction size 4096

Change 96993 on 2003/04/22 by ashishs@fl_ashishs_r400_win

added the TP_NoMipPointRound=1 to the dumps on and off

Change 96987 on 2003/04/22 by ashishs@fl_ashishs_r400_win

adding the TP_NoMipPointRound.reg to the all dumps on and off

Change 96983 on 2003/04/22 by ashishs@fl_ashishs_r400_win

need to set this registry key for some of the failing CL/VTE tests.
(r400vte_z_fmt_02,r400vte_combos_02,r400cl_point_rectangular_vtxXport_clip_01,r400cl_point_rectangular_vtxXport_actualClip_01)

Change 96954 on 2003/04/22 by omesh@omesh_r400_linux_marlboro

Added 3 and 6 sample testcase (6 each) versions for the various triangle
sizes.

Change 96900 on 2003/04/22 by mdoggett@mdoggett_r400_linux_local

Added 4x16expand.

Change 96787 on 2003/04/21 by csampayo@fl_csampayo_r400

Initial chaeck-in of test using mova_floor in vtx shader

Change 96785 on 2003/04/21 by mdoggett@mdoggett_r400_linux_local

Added basic 64 and 128bpp formats. Also 32bpp that expand to 64bpp.

Change 96742 on 2003/04/21 by mkelly@fl_mkelly_r400_win_laptop

During Publish = "true" output vectors to t:/r400/regress instead of
the old fl_mkelly2 path...

Change 96738 on 2003/04/21 by mmang@mmang_crayola_linux_orl

Fixed bug in sq_ais_output.v related to address register write and
predication. Fixed a variety of tests to not use uninitialized gpr
or address registers. 2 tests still fail because of previous vector
scalar swizzle bug, 1 test still fails because of MOVA hardware bug, and
1 test still fails because of predicated address register write causes
XXXXXX which causes waterfalling to hang.

Change 96699 on 2003/04/21 by mkelly@fl_mkelly_r400_win_laptop

Test all 32 RTS boolean bits in the pixel shader...

Change 96676 on 2003/04/21 by csampayo@fl_csampayo_r400

Update to version 2.0

Change 96551 on 2003/04/19 by markf@markf_r400_lt_marlboro

Disabled mip packing in TC tests.

Change 96522 on 2003/04/18 by csampayo@fl_csampayo3

Updated to print pixel shader multiplier

Change 96513 on 2003/04/18 by csampayo@fl_csampayo3

Adjust pix shader constant from 765.1 to 765.01

Change 96512 on 2003/04/18 by mdoggett@mdoggett_r400_linux_local

Updated to corrected decompression and compression code.

Change 96496 on 2003/04/18 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint, first 16 RT bools with control flow and huge pixel shader...

Change 96481 on 2003/04/18 by ashishs@fl_ashishs_r400_win

oops forgot to make total cases = 170

Change 96479 on 2003/04/18 by omesh@omesh_r400_linux_marlboro

Changed 128x128 tests to be 0 to 127 rather than 0 to 128, as earlier
erroneously typed.

Change 96478 on 2003/04/18 by ashishs@fl_ashishs_r400_win

extending the test to 512 constants...

Change 96477 on 2003/04/18 by csampayo@fl_csampayo3

Some ALU constants reordering

Change 96462 on 2003/04/18 by georgev@devel_georgev_r400_lin2_marlboro_tott

Added no blend buffers.

Change 96451 on 2003/04/18 by ashishs@fl_ashishs_r400_win

checking in the test so that carlos could further debug it

Change 96450 on 2003/04/18 by jhoule@jhoule_r400_win_lt

Adding dimmed version for degamma sanity tests

Change 96433 on 2003/04/18 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint on a simple test...

Change 96407 on 2003/04/18 by ashishs@fl_ashishs_r400_win

added few mysteriously missing tests...

Change 96387 on 2003/04/18 by csampayo@fl_csampayo3

Initial check-in

Change 96365 on 2003/04/18 by mkelly@fl_mkelly_r400_win_laptop

Enhance the hard code option -w to handle c:/proj/crayola client root.

Change 96281 on 2003/04/17 by jhoule@jhoule_r400_win_lt

Adding tp_ch_blend testbench app code

Change 96264 on 2003/04/17 by mdoggett@mdoggett_r400_linux_local

Added formats 45, 46, 47, 48.

Change 96221 on 2003/04/17 by jhoule@jhoule_r400_win_lt

Added RTL generator for ABCD table.
The emulator can now completely generate all tables as well as input index.
Still missing index clamping in RTL generator (for invalid sample IDs).

Fixed wrong ABCD float format.

Added missing assignment which gave garbage values when SAMPLE_LOCATION=Center.

Change 96217 on 2003/04/17 by omesh@omesh_r400_linux_marlboro

Added 8 other minor missing testcase scenarios.

Change 96080 on 2003/04/16 by csampayo@fl_csampayo_r400

Added Vineet

Change 96079 on 2003/04/16 by omesh@omesh_r400_linux_marlboro

Added another 12 sub_* testcases to cover non tile aligned clear areas
to test the combination of tile clear logic as well as non-fully covered
tile pixels falling through the shader pipe. These will show up on the
day after tommorow's nightly regression, as I just missed tonight's
regression.

Change 96068 on 2003/04/16 by omesh@omesh_r400_linux_marlboro

Added 13 Stencil Fast Clear and Expand testcases.

Change 96050 on 2003/04/16 by dougd@dougd_r400_linux_marlboro

added many test cases for multi context testing of each of the constant stores

Change 95986 on 2003/04/16 by ashishs@fl_ashishs_r400_win

finalising the script. Ready for use currently.

Change 95970 on 2003/04/16 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 95958 on 2003/04/16 by ashishs@fl_ashishs_r400_win

updated to have the correct formatting

Change 95957 on 2003/04/16 by omesh@omesh_r400_linux_marlboro

Added a filler testcase to take the place of an earlier misnamed
testcase. This new one has value too....

Change 95954 on 2003/04/16 by ashishs@fl_ashishs_r400_win

updated some groups

Change 95913 on 2003/04/16 by ashishs@fl_ashishs_r400_win

corrected the usage details for the script

Change 95888 on 2003/04/16 by ashishs@fl_ashishs_r400_win

script now ready for email with -e option. Please see usage for more details.

Change 95887 on 2003/04/16 by ashishs@fl_ashishs_r400_win

corrected script to have each line of the "email body" on a new line

Change 95874 on 2003/04/16 by omesh@omesh_r400_linux_marlboro

Changed the misleading 64x32 testcase names to 32x64 and also fixed some warnings.

Change 95861 on 2003/04/16 by jhoule@jhoule_r400_win_lt

Renamed into example.in

Change 95860 on 2003/04/16 by jhoule@jhoule_r400_win_lt

Renamed bug.in into example.in

Change 95848 on 2003/04/16 by ashishs@fl_ashishs_r400_win

updated to have correct paths for all the files (independent of the user's machine config, but depends on r400 environment config)

Change 95775 on 2003/04/15 by ashishs@fl_ashishs_r400_win

adding the base module for the email script

Change 95772 on 2003/04/15 by ashishs@fl_ashishs_r400_win

utility for sending emails from other applications. Please review readme if you would like to use it for your own application.

Change 95710 on 2003/04/15 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 95665 on 2003/04/15 by mkelly@fl_mkelly_r400_win_laptop

Add option -e for email.

Change 95662 on 2003/04/15 by omesh@omesh_r400_linux_marlboro

Split all SMASK accumulation tests into 2 groups: hier_stencil enabled, and hier_stencil disabled (r400rb_stencil_compression.cpp). All the Hi Stencil enabled tests can only run on the GC level testbench and the Hi Stencil disabled tests will run on the RBRC testbench. I will update the .rg nightly rbrc regression file soon....

Change 95556 on 2003/04/14 by omesh@omesh_r400_linux_marlboro

Changed test to use an explicit Fragment Data fill operation, recently added to Primlib by Kevin. Also used Larry's routine to calculate Slice Size, which enforces the 4KB slice boundary alignment.
I chose not to use the higher level multisample Primlib functions to preserve clarity of the test (enumarating all multisample registers set, to make the test more readable).

Change 95546 on 2003/04/14 by smoss@smoss_crayola_linux_orl

update

Change 95544 on 2003/04/14 by markf@markf_r400_lt_marlboro

Made MSAA test use minimum surface size of 64x64

Change 95539 on 2003/04/14 by smoss@smoss_crayola_linux_orl

update

Change 95536 on 2003/04/14 by smoss@smoss_crayola_linux_orl

more updates

Change 95520 on 2003/04/14 by jayw@jayw_r400_linux_marlboro

Added John's scripts.

Change 95492 on 2003/04/14 by omesh@omesh_r400_linux_marlboro

Changed test to use an explicit Fragment Data fill operation, recently added to Primlib by Kevin. Also used Larry's routine to calculate Slice Size, which enforces the 4KB slice boundary alignment.
Checked many (but not all) earlier asserting random testcases which were asserting due to the 4KB alignment problem and they now don't assert.

Change 95408 on 2003/04/14 by markf@markf_r400_lt_marlboro

Simple texture performance test

Change 95208 on 2003/04/11 by csampayo@fl_csampayo2_r400

Updated to use new SX MULTIPASS feature

Change 95194 on 2003/04/11 by ashishs@fl_ashishs_r400_win

removed the unsetting of env variable since being unset in sync.tcsh

Change 95191 on 2003/04/11 by ashishs@fl_ashishs_r400_win

unsetting the variable ...

Change 95189 on 2003/04/11 by jayw@jayw_r400_linux_marlboro

Fixed missing 'not' for Alpha Saturate.

Change 95185 on 2003/04/11 by mkelly@fl_mkelly_r400_win_laptop

Test all 32 booleans in RTS pixel shader...

Change 95168 on 2003/04/11 by mdoggett@mdoggett_r400_linux_local

Fixed 16_EXPAND bugs.
Stdout debug only printed if -debug flag used on commandline.

Change 95157 on 2003/04/11 by smoss@smoss_crayola_linux_orl

update

Change 95154 on 2003/04/11 by smoss@smoss_crayola_linux_orl

update

Change 95148 on 2003/04/11 by smoss@smoss_crayola_linux_orl

rev 1

Change 95140 on 2003/04/11 by csampayo@fl_csampayo_r400

Add sc

Change 95136 on 2003/04/11 by ashishs@fl_ashishs_r400_win

changed file names, hence deleted old

Change 95135 on 2003/04/11 by ashishs@fl_ashishs_r400_win

files for auto sync make regression on windows. See readme file for more details

Change 95109 on 2003/04/11 by mkelly@fl_mkelly_r400_win_laptop

Test smallest Z offset and scale to produce a discernable difference in the Zbuffer (1 lsb).

Change 95069 on 2003/04/10 by ashishs@fl_ashishs_r400_win

updated scripts so that doesnt require any environment variable to be set...

Change 95001 on 2003/04/10 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 94987 on 2003/04/10 by omesh@omesh_r400_linux_marlboro

Checking in the other 1536 testcases.... I visually verified a few on the emulator. Some look wrong on the emulator FB
output. Will file bugs after consultation with Larry.

Change 94977 on 2003/04/10 by ashishs@fl_ashishs_r400_win

updated test_list

Change 94888 on 2003/04/10 by ashishs@fl_ashishs_r400_win

Files added for auto sync, make , regression and emailing results.

For these scripts to work the following environment variables have to be set permanently in the Windows system.
AUTO = false // has to be set false all the time
SC_REGRESS = false
SQ_REGRESS = false
VGT_REGRESS = false
SU_REGRESS = false
CL_REGRESS = false
VTE_REGRESS = false
The other env var's could be set to true or false depending on the user's choice. Also these var's are used only in auto mode so the sync.tcsh script isn't affected.

==========================
sync.tcsh from Steve Moss :
==========================

This script when run manually will prompt for the blocks whose regression is to be performed. On setting value of those blocks true or false, the script will sync to the top of the tree, make build and then perform the regressions of the blocks that were selected.

When the script is run in auto mode, the script performs the same task of sync, make and regressions but the only difference being that it performs the regressions of the blocks, whose value is set in the environment variable by the user. So please setup the environment variable accordingly so that everyday only those regressions will be performed.

Currently the script only sync's the following dir's : cmn_lib,emu_lib,test_lib . On request more could be added.

=============
auto.bat
=============

This is a batch file for auto sync, make, regress and email the report. This batch file should be started by the Windows scheduler at the time of your preference, preferably at night.
Following are the steps that you need to carry out for this to work.
1) You will need a sync.tcsh in the pa_regress directory. See pa_regress dir in test_lib for more details.
2) Set an environment variable "auto" as "false" using Windows Control Panel, System, Environment Variable.
2) You will also need to modify your .tcshrc file so that , if the value of the env variable auto is true you can bypass the prompt and source the sync.tcsh as mentioned in step 1

This batch file is used to start the auto sync, make and regress.

Change 94879 on 2003/04/10 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 94869 on 2003/04/10 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 94819 on 2003/04/09 by smoss@smoss_crayola_linux_orl

added another dump

Change 94813 on 2003/04/09 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 94664 on 2003/04/09 by mdoggett@mdoggett_r400_linux_local

Added format 43 (8bpp interlace). Fixed inputtcd.txt to have sector address in high bits of 4 bit sector value.

Change 94520 on 2003/04/08 by ashishs@fl_ashishs_r400_win

testing address register

Change 94509 on 2003/04/08 by mkelly@fl_mkelly_r400_win_laptop

If only test name exists in test_list_parameterized, then run all DEFINE_TEST_CASE test cases in test .cpp file

Change 94460 on 2003/04/08 by omesh@omesh_r400_linux_marlboro

Fixed Bugzilla#1664. Fixed test by using the max_sample_dist table, as used earlier for the r400rb_msaa tests. Verified that sample#2 of pixel (2,3) was broken before the fix and was fixed after the fix, as verified by the DumpView image produced by the emulator (ignoring tile buffer file). This fix also fixed some missing sample#3 pixels.

Change 94412 on 2003/04/08 by smoss@smoss_crayola_linux_orl

update

Change 94360 on 2003/04/07 by omesh@omesh_r400_linux_marlboro

Added 25% of intended testcases. Will add these to regression too.

Change 94324 on 2003/04/07 by ashishs@fl_ashishs_r400_win

testing address register with different set of data

Change 94233 on 2003/04/07 by ashishs@fl_ashishs_r400_win

testing address instruction

Change 94230 on 2003/04/07 by omesh@omesh_r400_linux_marlboro

Fixed a stupid problem with the test.

Change 94213 on 2003/04/07 by mdoggett@mdoggett_r400_linux_local

Fixed degamma bugs. Degamma channel bits not set correctly. (uint8) cast truncated uint16 values.

Change 94170 on 2003/04/07 by smoss@smoss_crayola_linux_orl

bunch of dumps

Change 94143 on 2003/04/06 by ashishs@fl_ashishs_r400_win

updated

Change 94035 on 2003/04/04 by omesh@omesh_r400_linux_marlboro

A compiler error involving enums was not being caught on Linux and this was causing the emulator to hang when it was running the test!!
Test fixed.... and now runs on the emulator.

Change 94020 on 2003/04/04 by jhoule@jhoule_r400_win_lt

Fixed missing inputs (register memory). Appropriately modified the writeInput function in the tp_lod_aniso_parser.

Added the standalone testbench which was previously missing.

Change 93992 on 2003/04/04 by omesh@omesh_r400_linux_marlboro

Added Hi Stencil tests. Finished most of the testing logic, just need to populate extensive testcases. I may use a script to generate this.
The "standard" testcase seems to be hanging the emulator.

Change 93962 on 2003/04/04 by jhoule@jhoule_r400_win_lt

tp_lod_aniso dedicated testbench.

Change 93948 on 2003/04/04 by mdoggett@mdoggett_r400_linux_local

Added YUV, 32a8, 32a88

Change 93940 on 2003/04/04 by markf@markf_r400_linux_marlboro

Fixed random TC tests

Change 93873 on 2003/04/04 by ashishs@fl_ashishs_r400_win

corrected the shader to a much better simpler algorithm for LIT

Change 93777 on 2003/04/03 by ashishs@fl_ashishs_r400_win

correted vte setup error, also added some documentation inside the shader

Change 93750 on 2003/04/03 by ashishs@fl_ashishs_r400_win

implemented LIT instruction in the shader (implemented using subroutine can be reused). still some modifications needed...

Change 93610 on 2003/04/03 by ashishs@fl_ashishs_r400_win

testing DST instruction

Change 93593 on 2003/04/03 by mdoggett@mdoggett_r400_linux_local

Added cycle to commented input format line.

Change 93589 on 2003/04/03 by mdoggett@mdoggett_r400_win_platypus

Added DXT1 2D tiled and linear.

Change 93550 on 2003/04/02 by markf@markf_r400_lt_marlboro

Constrained TC randoms to subset of formats for now

Change 93547 on 2003/04/02 by markf@markf_r400_lt_marlboro

Added 2D tiled 1bpp format

Change 93478 on 2003/04/02 by llefebvr@llefebvr_r400_emu_montreal

This is yet another case where not enough GPRs were reserved. Vineet please be extra carefull with these they take forever to debug. It was in the pass 1 vertex shader where 10 gprs were alloc. but 16 were used. Remove the GPR line to allow the assembler to automatically detect the number of GPRs this is a lot safer. Test works fine now.

Change 93463 on 2003/04/02 by ashishs@fl_ashishs_r400_win

testing call instruction. test gets position, color and 8 textures data from the constant memory, pointer to constant memory is passed to the vertex shader which extracts the data and exports it. Test has 2 subroutines, one for fetching the data from the constant memory into the internal registers and the second is to output the data.

Change 93455 on 2003/04/02 by markf@markf_r400_lt_marlboro

Switched filter to bilinear in TC video format tests

Change 93449 on 2003/04/02 by markf@markf_r400_lt_marlboro

Added TC tests for texture formats used in video (ex. interlaced)

Change 93448 on 2003/04/02 by ashishs@fl_ashishs_r400_win

testing call instruction. Test with 64 triangle list prim packets, vertex shader with 16 call instructions and each subroutine incrementing the value of x or y, thereby toggling between the 16 different subroutines to match the r300 image.

Change 93446 on 2003/04/02 by ashishs@fl_ashishs_r400_win

testing call instruction. Test having 256 triangle_list primitive packets , with the vertex shader having 16 call instructions and sequentially calling each subroutine in way that the input is transferred to output unchanged.

Change 93213 on 2003/04/01 by ashishs@fl_ashishs_r400_win

The test has 16 jump instructions, doing various calculations on input while jumping through various addresses but still keeping output data same as input. The input data is stored in the contsant memory and pointer to that is passed to the vertex shader.

Change 93168 on 2003/04/01 by ashishs@fl_ashishs_r400_win

testing JMP instruction. vertex shader having 16 jumps and randomly switching booleans thereby controlling jump addresses.

Change 93158 on 2003/04/01 by markf@markf_r400_lt_marlboro

TC 1bpp format test

Change 93147 on 2003/04/01 by ashishs@fl_ashishs_r400_win

added headers to all the shaders

Change 93128 on 2003/04/01 by markf@markf_r400_lt_marlboro

Reduced # of random formats in tc_random.cpp

Change 93112 on 2003/04/01 by markf@markf_r400_lt_marlboro

Updated shaders in TC tests to work w/ random formats

Change 93107 on 2003/04/01 by mdoggett@mdoggett_r400_win_platypus

example input file for tcd standalone testbench

Change 93103 on 2003/04/01 by mdoggett@mdoggett_r400_win_platypus

TCD standalone emulator executable.

Change 93101 on 2003/04/01 by ashishs@fl_ashishs_r400_win

testing JMP instruction. 256 packets of a simple small triangle processed thru the vertex shader each time using 16 jump addresses.

Change 93034 on 2003/03/31 by markf@markf_r400_lt_marlboro

Adding TC randoms

Change 92981 on 2003/03/31 by ashishs@fl_ashishs_r400_win

test for LOOP instruction. the test has 7 loops and fetches all of the vertex data from the constant memory using pointers to the constant memory address.

Change 92956 on 2003/03/31 by markf@markf_r400_lt_marlboro

Added simple endian test for TC

Change 92954 on 2003/03/31 by markf@markf_r400_lt_marlboro

Adding simple Vfetch format tests

Change 92931 on 2003/03/31 by jhoule@jhoule_r400_win_lt

uber_map.h:
----
setMapType now kills dimension sizes in order to set correct packing behavior in UberChain::packLastLevels.

Added various downsample_* routines (including 3D ones). Still missing odd-sized ones though.
This should fix 1D/2D/3D mipmap generation, as long as sizes are even.

tp_simple_mip_1d.cpp:

Changed buildLevel's height from 1 to 0, since this now correctly works in the UberMap.

Change 92929 on 2003/03/31 by abeaudin@abeaudin_r400_win_marlboro

The test were writing the texture into the frame buffer twice.
I removed the write to the start of the framebuffer where the image
was also being drawn

Change 92849 on 2003/03/31 by ashishs@fl_ashishs_r400_win

updated

Change 92837 on 2003/03/31 by ashishs@fl_ashishs_r400_win

changed to compile on linux

Change 92823 on 2003/03/31 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 92696 on 2003/03/28 by ashishs@fl_ashishs_r400_win

testing loop instruction

Change 92692 on 2003/03/28 by markf@markf_r400_lt_marlboro

More cubic env map tests

Change 92687 on 2003/03/28 by ygiang@ygiang_r400_linux_marlboro

fixed: default params values for tests result: min instructions calculation is corrected

Change 92656 on 2003/03/28 by jayw@jayw_r400_linux_marlboro

New AB fixes for avoiding neg zero out of blender.

Change 92646 on 2003/03/28 by ashishs@fl_ashishs_r400_win

testing loop instruction

Change 92634 on 2003/03/28 by llefebvr@llefebvre_laptop_r400_emu

In this test Z is exported in R1 from the vertex shader. Pixel shader was incorectly reading it from R0. Hence the missmatch. I fixed the shader and now the test works fine.

Change 92632 on 2003/03/28 by markf@markf_r400_lt_marlboro

Cubic mipmpapped tests for TC

Change 92612 on 2003/03/28 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint..

Change 92583 on 2003/03/28 by mkelly@fl_mkelly_r400_win_laptop

Update RT state for pixel shader...

Change 92581 on 2003/03/28 by omesh@omesh_r400_linux_marlboro_only_devel

Allowed each testcase to customize screen space size, to allow for
minimal FB file sizes.

Change 92567 on 2003/03/28 by markf@markf_r400_lt_marlboro

Added cubic environment map tests for TC

Change 92564 on 2003/03/28 by smoss@smoss_crayola_linux_orl

added sq stuff

Change 92560 on 2003/03/28 by ashishs@fl_ashishs_r400_win

testing some instructions together

Change 92430 on 2003/03/27 by ygiang@ygiang_r400_linux_marlboro

fixed: random test time out and number of random instructions

Change 92422 on 2003/03/27 by jayw@jayw_r400_linux_marlboro

-j 10

Change 92414 on 2003/03/27 by mkelly@fl_mkelly_r400_win_laptop

Two important tests for regress_e

Change 92329 on 2003/03/27 by kevino@kevino_r400_win_marlboro

Added formats 2_10_10_10, 10_11_11, and 11_11_10 to check_tfc_overrides code to turn off filtering in tp_multitexture_01.h

Change 92282 on 2003/03/26 by markf@markf_r400_lt_marlboro

Added volume map tests

Change 92216 on 2003/03/26 by omesh@omesh_r400_linux_marlboro_only_devel

Added other triangle sizes (quad coverage, multiple quad coverage, etc.)
for 2 and 4 samples.

Change 92213 on 2003/03/26 by omesh@omesh_r400_linux_marlboro_only_devel

Added some basic tests for 2 sample and 4 sample modes.

Change 92193 on 2003/03/26 by kryan@kryan_r400_win_marlboro_XP

Edit test to avoid compilation error on Windows since cannot

have auto variable for first index of multi-dimensional array.

Change 92118 on 2003/03/25 by markf@markf_r400_lt_marlboro

Updated texture_manager.cpp to do gradient fills in the buildlevel function. Added some utility functions to tconst.cpp / tconst.h. Added several tc tests.

Change 92076 on 2003/03/25 by ashishs@fl_ashishs_r400_win

testing SETGE inbstruction

Change 92057 on 2003/03/25 by ashishs@fl_ashishs_r400_win

initial checkin for testing muladd instruction

Change 92049 on 2003/03/25 by ygiang@ygiang_r400_linux_marlboro fixed: random shaders test

Change 92029 on 2003/03/25 by markf@markf_r400_lt_marlboro

Fixed HiZ enable

Change 91967 on 2003/03/25 by jhoule@jhoule_r400_win_lt

Added allocate/fillColor/genCheckerBoard in tp_uber_test.
Added appropriate support functions to UberMap and UberChain.
Updated tp_uber_test to enable specifying those.

Change 91881 on 2003/03/24 by csampayo@fl_csampayo_r400

Update to not use w component of input

Change 91839 on 2003/03/24 by omesh@omesh_r400_linux_marlboro_only_devel

Added several directed color channel mask tests in combo with
swap_lowblue (Apparently this combo was found to be bugged in the random
regressions on the emulator)

Change 91819 on 2003/03/24 by ashishs@fl_ashishs_r400_win

testing MAX instruction

Change 91814 on 2003/03/24 by ashishs@fl_ashishs_r400_win

testing MIN instruction

Change 91802 on 2003/03/24 by ashishs@fl_ashishs_r400_win

testing fract instruction

Change 91633 on 2003/03/21 by ashishs@fl_ashishs_r400_win

plotting points to get logarithmic curve

Change 91627 on 2003/03/21 by ashishs@fl_ashishs_r400_win

plotting points to get an exponential curve

Change 91611 on 2003/03/21 by markf@markf_r400_linux_marlboro

Simple texture test for formats

Change 91585 on 2003/03/21 by llefebvr@llefebvr_r400_emu_montreal

Fixing wrong aluId number in arbiter.

Fixing mem exports some more.

Change 91564 on 2003/03/21 by ashishs@fl_ashishs_r400_win

test for LOG INSTRUCTION

Change 91503 on 2003/03/21 by jhoule@jhoule_r400_ma-jhoule-linux

Added -lstdc++ to ULLIBS

Change 91486 on 2003/03/21 by lseiler@lseiler_r400_win_marlboro2

Depth buffering was disabled for the pred_ez test case, which exports depth.

Change 91479 on 2003/03/21 by markf@markf_r400_win_marlboro

random shader test

Change 91438 on 2003/03/20 by csampayo@fl_csampayo3

VGT output path stress tests

Change 91302 on 2003/03/20 by jhoule@jhoule_r400_win_lt

tp_lod_deriv dedicated testbench modifications.

emu_lib/model/tp:
- Added tp_lod_deriv.{h|cpp} and tp_lod_deriv_parser.{h|cpp}
- Fixed bugs in the tp_lod_corrector (precision loss forced A into BCD)
- Added BD-AC term in ABCD class
- Cleaned up print routines in the corrector
- Added Float_32b, Float_16b, and Float_24b in tp_types.h (used by Gradients)

test_lib/src/chip/gfx/tp/standalone/tp_lod_deriv:
- Links statically with gfx_model.lib in order to remove exported symbol issues
- Options parsing enabling dumping of the ABCD terms

Note: this subblock is NOT yet integrated in tp.cpp.

Change 91276 on 2003/03/20 by jhoule@jhoule_r400_ma-jhoule-linux

Adding simple test to check TP_TC testbench

Change 91174 on 2003/03/20 by mkelly@fl_mkelly_r400_win_laptop

Updated based on advise from Harry...
I recommend setting the registry key to speed up the validation

environment:
[HKEY_LOCAL_MACHINE\SOFTWARE\ATI Technologies\Debug]
"pm4MicroLoadDisable"=dword:00000001

This key will allow the CP Microcode to be loaded via a back
door load rather than many register writes.

Change 91034 on 2003/03/19 by ashishs@fl_ashishs_r400_win

Testing RECIPSQ_IEEE instruction

Change 91028 on 2003/03/19 by jhoule@jhoule_r400_win_lt

Changed VFetch to ready __X1 instead of __Z1 when using FMT_32_FLOAT.
Better test citizen for RTL code, since HiColor channel replication is not a HW requirement.

Change 90991 on 2003/03/19 by omesh@omesh_r400_linux_marlboro

Added fragment data surface and initialized it (0xab......) within the address space
of the color buffer surface.

Change 90985 on 2003/03/19 by ashishs@fl_ashishs_r400_win

Testing: RECIP instruction
This program moves the reciprocal of the input to the output using the RCP instruction with swizzle controls and write enables

Change 90925 on 2003/03/19 by ashishs@fl_ashishs_r400_win

Testing MUL instruction using input memory, constant memory and output This program loads the constant memory with random and constants using the loaded constants and the MUL instruction it massages the position, textures, colors.The results are output to the output buffers and should be the same as passing input thru to output

Change 90907 on 2003/03/19 by omesh@omesh_r400_linux_marlboro

Changed Destination Compare and Color Mask Compare programming to be
non-replicating of values (Higher order 0 filled instead), in accordance
with spec change. Haven't verified that it does/ does not assert with
new emulator changes.

Change 90898 on 2003/03/19 by ashishs@fl_ashishs_r400_win

Testing Move input to output with swizzle and write enables .This shader moves the input to the output using swizzle controls and write enables

Change 90883 on 2003/03/19 by mkelly@fl_mkelly_r400_win_laptop

SC debug register coverage...

Change 90769 on 2003/03/18 by ashishs@fl_ashishs_r400_win

Testing DOT3 instruction using input memory, constant memory and output buffers
This program loads the constant memory with matrices and constants
using the loaded matrices it rotates the position and textures input
vectors and scales the input color, fog and The results are output
to the output buffers and should be the same as passing input thru to output

Change 90744 on 2003/03/18 by mkelly@fl_mkelly_r400_win_laptop

Test para_enable bit, update register coverage...

Change 90741 on 2003/03/18 by ashishs@fl_ashishs_r400_win

Testing DP4 instruction using input memory, constant memory and output buffers
This program loads the constant memory with matrices and constants
using the loaded matrices it rotates the position and textures input
vectors and scales the input color. The results are output to the output
buffers and should be the same as passing input thru to output .

Change 90713 on 2003/03/18 by ashishs@fl_ashishs_r400_win

same as r400sq_add_01 but with different set of constant data such that the operations lie within the ieee floating point limits

Change 90711 on 2003/03/18 by ashishs@fl_ashishs_r400_win

This test is intended to check the ADD instruction. After adding and subtractiong random constants to the input vertex data it moves the final result to the output buffers, effectively passing inputs to outputs.
Single triangle with XYZW, 4 colors and 6 textures.
The constants are setup in such a way that the register operations go in and out of the limits of ieee floating point limits during run time.

Change 90698 on 2003/03/18 by omesh@omesh_r400_linux_marlboro

Fixed the test problem involving max_sample_dist. The test now produces
the missing 7th sample (0-7 samples).

Change 90685 on 2003/03/18 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 90673 on 2003/03/18 by jayw@jayw_r400_linux_marlboro new regress scripts

Change 90655 on 2003/03/18 by omesh@omesh_r400_linux_marlboro

Added a fan1_8samp_color8888_bug testcase that scissors the missing
sample pixel in question. This will aid the SC people to debug the
symptom better.

Change 90613 on 2003/03/17 by csampayo@fl_csampayo_r400

Control flow, predicate and multi-context and multi-prim test. Upfdated test_list accordingly

Change 90525 on 2003/03/17 by markf@markf_r400_lt_marlboro

Added some HiZ tests

Change 90501 on 2003/03/17 by smoss@smoss_crayola_linux_orl

lsf makefile for sq

Change 90475 on 2003/03/17 by csampayo@fl_csampayo_r400

Adusted timeout setting in cpp (+some cleanup), enabled LOOP 0 in vertex shader and updated some sub-routines for co-issue instructions

Change 90373 on 2003/03/15 by ygiang@ygiang_r400_pv2_marlboro

fixed: cube random test
added: vector random tests for sp

Change 90309 on 2003/03/14 by kmeekins@kmeekins_r400_win

Clamped the FIFO size to use a minimum depth of 8.

Change 90251 on 2003/03/14 by mkelly@fl_mkelly_r400_win_laptop

sc_sp centers/centroids parameters 13, 14, 15

Change 90244 on 2003/03/14 by jhoule@jhoule_r400_win_marlboro

Contains 3 forms of VFetches:
- FMT_32_FLOAT
- FMT_32_32_FLOAT + FMT_32_FLOAT
- FMT_32_32_32_32_FLOAT

Current set to the middle one.

Change 90230 on 2003/03/14 by mkelly@fl_mkelly_r400_win_laptop

8 MSAA masked sample 1 with resolve...

Change 90221 on 2003/03/14 by csampayo@fl_csampayo_r400

Update for co-issue instructions

Change 90179 on 2003/03/14 by csampayo@fl_csampayo2_r400

Sort

Change 90164 on 2003/03/14 by ashishs@fl_ashishs_r400_win

corrected texture maps in the test to match R300 test

Change 90161 on 2003/03/14 by mkelly@fl_mkelly_r400_win_laptop

sc_sp sample control parameters 8 - 13

Change 90144 on 2003/03/14 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 90081 on 2003/03/13 by georgev@devel_georgev_r400_lin2_marlboro_tott

More texture registers.

Change 90069 on 2003/03/13 by jayw@jayw_r400_linux_marlboro

More LEDA.

Change 90054 on 2003/03/13 by georgev@devel_georgev_r400_lin2_marlboro_tott

Made texture bigger.

Change 90047 on 2003/03/13 by llefebvr@llefebvr_r400_emu_montreal

fixing test to use center instead of centroid on p2.

Change 90042 on 2003/03/13 by ashishs@fl_ashishs_r400_win

updated

Change 90039 on 2003/03/13 by ashishs@fl_ashishs_r400_win

This test processes a single triangle with input/output vertex data: xyzw0; colors 0-3 and textures 0-7
Two sets of vertex data are stored in constant memory and only pointers to them are set in vertex buffer.

The vertex shader program loaded has several loops jumps and calls. The test processes 16 packets
randomly selecting booleans so that one or the other set of vertex data is processed and loop controls
are also randomly set.

Change 90038 on 2003/03/13 by georgev@devel_georgev_r400_lin2_marlboro_tott

Fix /0 error.

Change 90035 on 2003/03/13 by mkelly@fl_mkelly_r400_win_laptop

Add dummy file for script

Change 90033 on 2003/03/13 by mkelly@fl_mkelly_r400_win_laptop

sc_sp sampling through interpolators, parameters 3 - 8

Change 90031 on 2003/03/13 by georgev@devel_georgev_r400_lin2_marlboro_tott

Added new tests to parameterized list.

Change 90011 on 2003/03/13 by georgev@devel_georgev_r400_lin2_marlboro_tott

Changed to do textures only for 1 test. It's faster that way.

Change 89996 on 2003/03/13 by mkelly@fl_mkelly_r400_win_laptop

Sampling methods on parameter 3...

Change 89992 on 2003/03/13 by jayw@jayw_r400_linux_marlboro

LEDA fixes.

Change 89989 on 2003/03/13 by mkelly@fl_mkelly_r400_win_laptop

Sample combinations on 2nd parameter in interpolators via sc_sp

Change 89951 on 2003/03/13 by mkelly@fl_mkelly_r400_win_laptop

Prim type detection on gpr position 0, and 9 - 15

Change 89904 on 2003/03/12 by georgev@devel_georgev_r400_lin2_marlboro_tott

Added last tests.

Change 89860 on 2003/03/12 by markf@markf_r400_lt_marlboro

Fixed depth clear value and stencil clear value for r400rb_zwave

Change 89829 on 2003/03/12 by ashishs@fl_ashishs_r400_win

added some VS and PS base and size control statements for random changes. commented for now , the test uses VS and PS loaded at 0 and 256 respectively and similarly for constants.

Change 89800 on 2003/03/12 by mkelly@fl_mkelly_r400_win_laptop

Primtype detection in the pixel shader, gpr positions 2 - 8

Change 89765 on 2003/03/12 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 89699 on 2003/03/12 by mkelly@fl_mkelly_r400_win_laptop

Check prim type in the shader for parameter gen pos 2

Change 89574 on 2003/03/11 by ashishs@fl_ashishs_r400_win

changed pix center to better match equivalent r300 test

Change 89570 on 2003/03/11 by mkelly@fl_mkelly_r400_win_laptop

Update comments

Change 89569 on 2003/03/11 by mkelly@fl_mkelly_r400_win_laptop

Check POLY, POINT, LINE prim type detection in SP on parameter 0

Change 89567 on 2003/03/11 by ashishs@fl_ashishs_r400_win

This test is intended to validate the ALU constant and instruction memory
auto wrapping function. The test processes 64 packets each with
one 8 primitive triangle list containing 9 vertices with input/output
vertex data: XYZW, 4 colors and 8 textures . For each packet the test
sends a new matrix to the constant memory so that the shader rotates
vertices differently from the others. The shader uses up all 256 entries
of the constant memory and the instruction memory size is 256. Also, for
each packet the conastant and code memory offsets are updated with random
values, as well as, the loading method is selected between 32/128 bits
randomly.
Currently the test hasnt been tested with loading shaders at different
offsets and thereby checking the wrapping of memory.

Change 89553 on 2003/03/11 by georgev@devel_georgev_r400_lin2_marlboro_tott

Changes for bug 1449. Still not done.

Change 89535 on 2003/03/11 by mkelly@fl_mkelly_r400_win_laptop

Test XY on parameters 11 - 15 and 0 for sc_sp interface

Change 89507 on 2003/03/11 by csampayo@fl_csampayo_r400

Update...

Change 89503 on 2003/03/11 by mkelly@fl_mkelly_r400_win_laptop

XY position from SC to SP on parameters 8 - 10

Change 89471 on 2003/03/11 by mkelly@fl_mkelly_r400_win_laptop

Validate XY position from SC to SP, parameters 2 - 7

Change 89419 on 2003/03/10 by jayw@jayw_r400_linux_marlboro

Fixes for very 16 bit srepeat and urepeat precision. hang fix, bad state for tiling, unknowns for rc cache signal

Change 89382 on 2003/03/10 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 89369 on 2003/03/10 by mkelly@fl_mkelly_r400_win_laptop

Export position as color 0 to the pixel shader to avoid SQ Deallocation Assert
on an empty parameter store on a Position Only vertex.

Change 89364 on 2003/03/10 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 89359 on 2003/03/10 by omesh@omesh_r400_linux_marlboro_only_devel

Increased framebuffer size for some larger testcases. Also added
RB_SURFACE_EXTENT (pitch) field programming. Also added some more
testcases.

Change 89343 on 2003/03/10 by mkelly@fl_mkelly_r400_win_laptop

Update test based on recommendation from Kevin to fix RB assert for pitch

Change 89328 on 2003/03/10 by mkelly@fl_mkelly_r400_win_laptop

Finish checking all parameters for face bit in pixel shader...

Change 89311 on 2003/03/10 by mkelly@fl_mkelly_r400_win_laptop

Back face bit check parameters 3 through 8

Change 89297 on 2003/03/10 by mkelly@fl_mkelly_r400_win_laptop

Back face bit check on parameter 2...

Change 89296 on 2003/03/10 by kryan@kryan_r400_win_marlboro_XP

- Modified test to generate the correct resolve surface for the case when HEIGHT > 32, specifically:

SURFACE_PITCH = 128

SURFACE_HEIGHT = 64

Had to add lines to the resolve_buffer() function to set

the RB_SURFACE_EXTENTS pitch and height fields.

- Also modified the render_state parameter to pass by reference (&render_state) rather than by value in the

resolve_buffer() function.

The test gfx/sc/r400sc_msaa_8_resolve_01.cpp should update its

resolve_buffer() function to match these new changes.

Change 89289 on 2003/03/10 by mkelly@fl_mkelly_r400_win_laptop

Changed test name, added to emu regression since it touches perf regs

Change 89285 on 2003/03/10 by mkelly@fl_mkelly_r400_win_laptop

Removed separate render state function for resolve and integrated it
into Main part of test using same render state throughout.

Change 89120 on 2003/03/07 by csampayo@fl_csampayo_r400

Try and add the right test this time.

Change 89118 on 2003/03/07 by csampayo@fl_csampayo_r400

New test checking single/dual vertex vectors of various sizes. Updated test_list and test tracker accordingly

Change 89106 on 2003/03/07 by georgev@devel_georgev_r400_lin2_marlboro_tott

Added some new tests for Vic and Doug.

Change 89093 on 2003/03/07 by omesh@omesh_r400_linux_marlboro_only_devel

Added some more basic multisample testcases that stress the fragment
cache.

Change 89060 on 2003/03/07 by smoss@smoss_crayola_linux_orl

queue modified to remove lfcs25

Change 89059 on 2003/03/07 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 89047 on 2003/03/07 by georgev@devel_georgev_r400_lin2_marlboro_tott

Fixed back jump labels.

Change 89037 on 2003/03/07 by llefebvr@llefebvr_r400_emu_montreal

fixing dumps and memory export test

Change 89032 on 2003/03/07 by csampayo@fl_csampayo_r400

Added pass-thru test with large (>64 indices) vertex vectors. Updated test_list and test tracker accordingly

Change 89025 on 2003/03/07 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 89020 on 2003/03/07 by llefebvr@llefebvr_r400_emu_montreal

bad GPR alloc on this test

Change 89016 on 2003/03/07 by mkelly@fl_mkelly_r400_win_laptop

Shader face bit check on parameter 2

Change 89008 on 2003/03/07 by csampayo@fl_csampayo_r400

Cleanup

Change 89006 on 2003/03/07 by mkelly@fl_mkelly_r400_win_laptop

Multi-Pass Pixel shader, segment 0 color exported to segment 1...

Change 89001 on 2003/03/07 by georgev@devel_georgev_r400_lin2_marlboro_tott

Added new tests, whacked on old ones.

Change 88929 on 2003/03/06 by vromaker@vromaker_r400_linux_marlboro

fix to CFS that prevents a new thread from entering when the thread ID
in the input pipe stage is different than the thread ID in the output pipe stage;
also changed triangle size to 150 for sq_tests test case pred_eq_vec

Change 88913 on 2003/03/06 by csampayo@fl_csampayo_r400

Adding mixed VGT 2/1 output vectors tests

Change 88912 on 2003/03/06 by ashishs@fl_ashishs_r400_win

deleted since added the test r400sq_16tex_flatShade_combo_allFlat_01

Change 88911 on 2003/03/06 by ashishs@fl_ashishs_r400_win

updated

Change 88904 on 2003/03/06 by georgev@devel_georgev_r400_lin2_marlboro_tott

Added max GPR multi context tests.

Change 88902 on 2003/03/06 by ashishs@fl_ashishs_r400_win

updated

Change 88901 on 2003/03/06 by ashishs@fl_ashishs_r400_win

flatShading all the parameters with ability to determine exact parameter failing using different color combinations

Change 88885 on 2003/03/06 by ashishs@fl_ashishs_r400_win

updated

Change 88877 on 2003/03/06 by dougd@dougd_r400_linux_marlboro

added " DEFINE_TEST_CASE_NAME(reg_indexing_max_gpr);" for the new test case

Change 88872 on 2003/03/06 by llefebvr@llefebvr_r400_emu_montreal

fixed shader.

Change 88867 on 2003/03/06 by ashishs@fl_ashishs_r400_win

adding test same as r400sq_16tex_flatShade_combo_first_01 with provoking vtx as LAST

Change 88865 on 2003/03/06 by ashishs@fl_ashishs_r400_win

changed file names by deleting the old files

Change 88864 on 2003/03/06 by ashishs@fl_ashishs_r400_win

SQ Interpolation test. This tests checks the "param_shade" field of the SQ_INTERPOLATOR_CNTL register. The 16 textures are transferred on 16 different parameters with the texture corners giving the vertex color. 64 different combinations (out of 2^16 possible) of the 16 bit param_shade register thereby having combination of flat as well as gouraud shading on different parameters with first vertex as the provoking vtx.

Change 88852 on 2003/03/06 by georgev@devel_georgev_r400_lin2_marlboro_tott

New test for GPR registers. Updates to MC test.

Change 88770 on 2003/03/06 by ashishs@fl_ashishs_r400_win

changed nan retain to default(false)

Change 88746 on 2003/03/06 by mkelly@fl_mkelly_r400_win_laptop

Completes initial check of prim type detection in the pixel shader
checking SQ POINT (r400sc_sp_sample_cntl_09), SQ LINE (r400sc_sp_sample_cntl_11)
and SQ POLY (this checkin).

Change 88740 on 2003/03/06 by mkelly@fl_mkelly_r400_win_laptop

Finalize test now that emu_lib bugs have been squashed...

Change 88691 on 2003/03/05 by ashishs@fl_ashishs_r400_win

forgot to add Part2

Change 88690 on 2003/03/05 by ashishs@fl_ashishs_r400_win

removed a test error. checked nan_retain bits with CLIPPING DISABLED.

if nan_retain is false for x,y,z,w the clipper discards the primitives having infinite data and the vte generates a null primitive for each of the discarded primitive, also if the vte generates infinity/nan data for the primitive (using scale/offset) primitive is clamped to -4k to 12k (as seen in ClipGa_alg.dmp)

if nan_retain is true for x,y,z,w the clipper retains the primitives even if they have nan/infinite data in the vertex, and the primitive is clamped in the vte which has nan/infinite data (nan/inf data passed from the clipper as well as generated by the vte using scale/offset)

Change 88687 on 2003/03/05 by llefebvr@llefebvr_r400_emu_montreal

Fixing +/-0 bug in emu and type setting.

Change 88665 on 2003/03/05 by vromaker@vromaker_r400_linux_marlboro

reduced default triangle size to 15, reduced multi_context triangle size to 5

Change 88659 on 2003/03/05 by jayw@jayw_r400_linux_marlboro

Bug fix for missing define.

Change 88634 on 2003/03/05 by mkelly@fl_mkelly_r400_win_laptop

Update shader

Change 88624 on 2003/03/05 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 88592 on 2003/03/05 by mkelly@fl_mkelly_r400_win_laptop

Update to demonstrate issue

Change 88588 on 2003/03/05 by mkelly@fl_mkelly_r400_win_laptop

Pixel shader primtype LINE detection from SC / SQ / SP

Change 88584 on 2003/03/05 by mkelly@fl_mkelly_r400_win_laptop

Comment update...

Change 88581 on 2003/03/05 by ygiang@ygiang_r400_pv2_marlboro

added: more sp cube tests

Change 88578 on 2003/03/05 by mkelly@fl_mkelly_r400_win_laptop

Zero out event initiator value in test before writing the event field

Change 88575 on 2003/03/05 by csampayo@fl_csampayo_r400

Do not index vector w component since, it does not get to SQ.

Change 88535 on 2003/03/05 by mkelly@fl_mkelly_r400_win_laptop

Gold for multipass IBs and SC loop signaling...

Change 88534 on 2003/03/05 by mkelly@fl_mkelly_r400_win_laptop

MultiPass Indirect Buffer multiple looping...

Change 88518 on 2003/03/05 by mkelly@fl_mkelly_r400_win_laptop

Change a comment...

Change 88517 on 2003/03/05 by smoss@smoss_crayola_linux_orl

update

Change 88515 on 2003/03/05 by mkelly@fl_mkelly_r400_win_laptop

MultiPass Indirect Buffer / SC pixel LOOP interaction / 2 Segment / 2 Pass

Change 88514 on 2003/03/05 by smoss@smoss_crayola_linux_orl

update

Change 88511 on 2003/03/05 by omesh@omesh_r400_linux_marlboro_only_devel

Added all the current changes for multisampling tests recently proved to be pass on h/w vs. emulator in the
basic multisample testcase.

Change 88508 on 2003/03/05 by mkelly@fl_mkelly_r400_win_laptop

POINT primtype detection in the pixel shader...

Change 88503 on 2003/03/05 by mkelly@fl_mkelly_r400_win_laptop

Update screen to 128x64 to show resolve dump bug...

Change 88445 on 2003/03/04 by csampayo@fl_csampayo_r400

More VGT pass-thru tests checking grouper data types

Change 88419 on 2003/03/04 by smoss@smoss_crayola_linux_orl

add lsf makefile

Change 88351 on 2003/03/04 by jhoule@jhoule_r400_win_marlboro

Better colors.
Values are now 0x00, 0x40, 0x80, 0xB0, 0xFF.

Change 88350 on 2003/03/04 by jhoule@jhoule_r400_win_marlboro

Added support for clipping in frame_buffer block.
Use clip_x, clip_y to specify the top left corner, and clip_w, clip_h to specify the size of the clipped area.

Change 88298 on 2003/03/04 by mkelly@fl_mkelly_r400_win_laptop

Set scissors to process exactly 2 pixel vectors

Change 88293 on 2003/03/04 by smoss@smoss_crayola_linux_orl

<Orlando Hardware Regression Results >

Change 88290 on 2003/03/04 by llefebvr@llefebvr_r400_emu_montreal

multiple state setting related issues with this test.

Change 88211 on 2003/03/03 by ashishs@fl_ashishs_r400_win

updated

Change 88187 on 2003/03/03 by ashishs@fl_ashishs_r400_win

commenetd out r400sq_plane.cpp

Change 88186 on 2003/03/03 by ashishs@fl_ashishs_r400_win

updated to include the SQ block

Change 88142 on 2003/03/03 by ashishs@fl_ashishs_r400_win

checkin for regress_e

Change 88123 on 2003/03/03 by kevino@kevino_r400_win_marlboro

Fixed test

Change 88118 on 2003/03/03 by mkelly@fl_mkelly_r400_win_laptop

Detect point/non-point primtype in shader

Change 88104 on 2003/03/03 by kevino@kevino_r400_linux_marlboro

a couple test fixes

Change 88076 on 2003/03/03 by kevino@kevino_r400_linux_marlboro

Tests compile (but not checked)

Change 88068 on 2003/03/03 by kevino@kevino_r400_win_marlboro

Added a couple simple highlat tests

Change 88039 on 2003/03/03 by smoss@smoss_crayola_linux_orl

added .h and .sp copy

Change 88036 on 2003/03/03 by georgev@devel_georgev_r400_lin2_marlboro_tott

Updated for test name changes.

Change 88028 on 2003/03/03 by omesh@omesh_r400_linux_marlboro_only_devel

Added background clear color register programming to make sure h/w flushes out the right color during
color expansion.

Change 88003 on 2003/03/03 by georgev@devel_georgev_r400_lin2_marlboro_tott

Changed test names for Vic.

Change 87844 on 2003/02/28 by jhoule@jhoule_r400_win_marlboro

Support for scissoring in frame_buffer block.

Change 87816 on 2003/02/28 by georgev@devel_georgev_r400_lin2_marlboro_tott

Added Franks test for MC stuff.

Change 87780 on 2003/02/28 by jhoule@jhoule_r400_win_marlboro

Missing texture for clamping tests.

Change 87769 on 2003/02/28 by ashishs@fl_ashishs_r400_win

updated to change the texture coords

Change 87765 on 2003/02/28 by mkelly@fl_mkelly_r400_win_laptop

You will need this vtx shader for the multi-pass test...

Change 87764 on 2003/02/28 by smoss@smoss_crayola_win

SU tests

Change 87761 on 2003/02/28 by mkelly@fl_mkelly_r400_win_laptop

Changed scissor so SC has more pixel vectors to process...

Change 87756 on 2003/02/28 by mkelly@fl_mkelly_r400_win_laptop

Getting closer on multi-pass IB / SC interaction....

Change 87750 on 2003/02/28 by smoss@smoss_crayola_win

added 0x8000 line

Change 87730 on 2003/02/28 by ashishs@fl_ashishs_r400_win

to test the SQ parameter wrapping for para 0 on S and T bits.

The test has 2 very simple cases with the first case being wrapped on S
and the second case being wrapped on T.

To demonstrate this 2 color band textures(one vertical and one horizontal color band)
are loaded in the texture memory. 2 Shaders are generated using the VFD such that
in 1 shader texture 0 is fetched using parameter 0 and in the second shader texture
1 is fetched using the same parameter 0.

Currently you can see that the texture is being just wrapped in T and the
parameter wrapping in S fails.

Change 87727 on 2003/02/28 by smoss@smoss_crayola_linux_orl

build test list is true

Change 87688 on 2003/02/28 by smoss@smoss_crayola_win

update

Change 87678 on 2003/02/28 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint on multi-pass indirect buffers with SC signaling...

Change 87677 on 2003/02/28 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 87651 on 2003/02/28 by csampayo@fl_csampayo2_r400

Renamed all multi-context tests

Change 87464 on 2003/02/27 by ashishs@fl_ashishs_r400_win

initial checkin

Change 87462 on 2003/02/27 by ashishs@fl_ashishs_r400_win

temporary checkin for test

Change 87440 on 2003/02/27 by mmang@mmang_crayola_linux_orl

Predicated parameter cache writes.

Change 87392 on 2003/02/27 by smoss@smoss_crayola_linux_orl

removed timestamp for unix

Change 87280 on 2003/02/27 by csampayo@fl_csampayo2_r400

Sorted list

Change 87279 on 2003/02/27 by mkelly@fl_mkelly_r400_win_laptop

Update geometry to black and white pinwheel...

Change 87252 on 2003/02/27 by mkelly@fl_mkelly_r400_win_laptop

Name changes for Steve...

Change 87248 on 2003/02/27 by mkelly@fl_mkelly_r400_win_laptop

Name changes for Steve...

Change 87247 on 2003/02/27 by mkelly@fl_mkelly_r400_win_laptop

Name changes for Steve...

Change 87175 on 2003/02/26 by csampayo@fl_csampayo_r400

Update test to use all 8 contexts, update test tracker accordingly

Change 87168 on 2003/02/26 by csampayo@fl_csampayo_r400

Add new VGT pass-thru tests

Change 87165 on 2003/02/26 by ashishs@fl_ashishs_r400_win

initial checkin for flat shading interpolation on all the 16 parameters each with a different texture and blending color at texture corners for flatShading

Change 87146 on 2003/02/26 by smoss@smoss_crayola_linux_orl

update

Change 87056 on 2003/02/26 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 87051 on 2003/02/26 by mkelly@fl_mkelly_r400_win_laptop

Msaa 8 with resolve

Change 87048 on 2003/02/26 by kryan@kryan_r400_win_marlboro_XP

Modified test to initialize background of Fragment Data portion of the Color Surface with fragment_background_color.

Change 87045 on 2003/02/26 by ygiang@ygiang_r400_pv2_marlboro

added:sq control flow test

Change 87018 on 2003/02/26 by georgev@devel_georgev_r400_lin2_marlboro_tott

Added fixes for windows loop counter bug.

Change 86896 on 2003/02/25 by smoss@smoss_crayola_linux_orl

update

Change 86822 on 2003/02/25 by ashishs@fl_ashishs_r400_win

checking flat shading on each respective parameter export , using 16 shaders for each export, for first as well as last vtx as provoking vtx.

Change 86793 on 2003/02/25 by georgev@devel_georgev_r400_lin2_marlboro_tott

Put in constant register writes for Doug.

Change 86782 on 2003/02/25 by markf@markf_r400_linux_marlboro

Added bilinear

Change 86728 on 2003/02/25 by mkelly@fl_mkelly_r400_win_laptop

Need PrimLib to add example useage, thank you.

Change 86721 on 2003/02/25 by mkelly@fl_mkelly_r400_win_laptop

Add cp_pfp_ib1.dmp

Change 86718 on 2003/02/25 by csampayo@fl_csampayo2_r400

Updated to change max mem to 128M

Change 86713 on 2003/02/25 by ashishs@fl_ashishs_r400_win

updated for r400sq_16tex_interp_combo_01

Change 86702 on 2003/02/25 by ashishs@fl_ashishs_r400_win

added GFX_IDLE_NO_FLUSH

Change 86695 on 2003/02/25 by ashishs@fl_ashishs_r400_win

generating 64 shaders, each unique in terms of combination of parameters it exports, thereby permuting selected few testcases out of the possible 2^16 combinations for parameter exports.

Change 86686 on 2003/02/25 by mkelly@fl_mkelly_r400_win_laptop

Fix test bug

Change 86661 on 2003/02/25 by mkelly@fl_mkelly_r400_win_laptop

Simple RTS for Christeen...

Change 86653 on 2003/02/25 by smoss@smoss_crayola_linux_orl

update

Change 86622 on 2003/02/24 by jhoule@jhoule_r400_win_marlboro

Added support for including tp_uber_test.

```
Recipe:
#define INCLUDE_TP_UBER_TEST
#include "tp_uber_test.cpp"

void  ParseTestInfo()
{
   //Read TESTINFO structure here
}
```

Added clamping test, which parses the TEST_CASE name to extract filename, clamping policies, clamping mode, border size, and filter type.

Change 86614 on 2003/02/24 by georgev@devel_georgev_r400_lin2_marlboro_tott

Fixed multi context tests to run quicker.

Change 86605 on 2003/02/24 by csampayo@fl_csampayo_r400

Adding pass-thru tests with 32 bit indices

Change 86601 on 2003/02/24 by ashishs@fl_ashishs_r400_win

correcting the test_list so that there are 21 tests for SQ non parameterised

Change 86593 on 2003/02/24 by csampayo@fl_csampayo2_r400

Remove duplicate test name

Change 86528 on 2003/02/24 by mkelly@fl_mkelly_r400_win_laptop

Multi-Pass Pixel shading test, needs CP Indirect Buffer Multi-Pass mode

Change 86472 on 2003/02/24 by jayw@jayw_r400_linux_marlboro_rbrc

Fixed SREPEAT negative values rounding.

Change 86426 on 2003/02/24 by omesh@omesh_r400_linux_marlboro_only_devel

Added an explicit No Blend programming, as emulator was recently
asserting.

Change 86423 on 2003/02/24 by ashishs@fl_ashishs_r400_win

updated

Change 86397 on 2003/02/24 by omesh@omesh_r400_linux_marlboro_only_devel

Added an explicit No Blend programming, as emulator was recently asserting.

Change 86391 on 2003/02/24 by georgev@devel_georgev_r400_lin2_marlboro_tott

First version to build all tests for diag.

Change 86351 on 2003/02/24 by mkelly@fl_mkelly_r400_win_laptop

Adding enhancement list for next project at end of script

Change 86147 on 2003/02/21 by jhoule@jhoule_r400_win_marlboro

Adding Clamping tests

Change 86146 on 2003/02/21 by jhoule@jhoule_r400_win_marlboro

Modified test to be able to specify texture information in multiple texture{} blocks, potentially spread across multiple files (using includes).

Change 86072 on 2003/02/21 by smoss@smoss_crayola_linux_orl

use root branch

Change 86069 on 2003/02/21 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 86067 on 2003/02/21 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 86064 on 2003/02/21 by mkelly@fl_mkelly_r400_win_laptop

Update gradient on Line RTS...

Change 86036 on 2003/02/21 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 86035 on 2003/02/21 by jayw@jayw_r400_linux_marlboro_rbrc

Fix for one pass fog ONLY precision down to 1.5.12

Change 86020 on 2003/02/21 by omesh@omesh_r400_linux_marlboro_only_devel

Changed test to use non symmetrical colors for vertices for more interesting testing.

Change 85994 on 2003/02/21 by mkelly@fl_mkelly_r400_win_laptop

Finish coverage of all x_dir y_dir walk directions for RTS prims

Change 85982 on 2003/02/21 by georgev@devel_georgev_r400_lin2_marlboro_tott

Made buffers indirect.

Change 85930 on 2003/02/21 by omesh@omesh_r400_linux_marlboro_only_devel

Fixed bug in test (color buffer was allocated AFTER the tile buffer, which resulted in an overlap).
Verified that Primlib now produces a non zero and sensible Tile Buffer Base Address.

Change 85917 on 2003/02/21 by ashishs@fl_ashishs_r400_win

added new tests using flat shading in the older tests..

Change 85915 on 2003/02/21 by jayw@jayw_r400_linux_marlboro

added DIFF.

Change 85891 on 2003/02/21 by ashishs@fl_ashishs_r400_win

changing test names

Change 85886 on 2003/02/21 by mkelly@fl_mkelly_r400_win_laptop

Add Larry's latest RB change control, default to 0
User must set it to 1 to get old RB blender

Change 85871 on 2003/02/21 by mkelly@fl_mkelly_r400_win_laptop

Add RB setup, extend timeout...

Change 85868 on 2003/02/21 by mkelly@fl_mkelly_r400_win_laptop

Extend timeout

Change 85867 on 2003/02/21 by mkelly@fl_mkelly_r400_win_laptop

Extend idle timeout, add RB setup for MSAA, disable JSS when MSAA = true,
set msaa num samples = 0 when JSS = true.

Change 85774 on 2003/02/20 by csampayo@fl_csampayo_r400

Adding point size clamping tests

Change 85752 on 2003/02/20 by lseiler@lseiler_r400_win_marlboro2

Fix RB blender precision problem and release updated golden images for regress_e

Change 85711 on 2003/02/20 by georgev@devel_georgev_r400_lin2_marlboro_tott

Added predicated Z test (first revision).

Change 85667 on 2003/02/20 by smoss@smoss_crayola_linux_orl

removed illegal .write tests

Change 85628 on 2003/02/20 by omesh@omesh_r400_linux_marlboro_only_devel

Fixed a typo in the test that was causing some testcases to assert. This
should fix about 4 assertions.

Change 85626 on 2003/02/20 by mkelly@fl_mkelly_r400_win_laptop

Update..

Change 85578 on 2003/02/20 by smoss@smoss_crayola_win

only look at rd_r file for read tests

Change 85577 on 2003/02/20 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 85558 on 2003/02/20 by mkelly@fl_mkelly_r400_win_laptop

Increased idle timeout, taking longer now that stipple is reset at
each primitive.

Change 85502 on 2003/02/20 by smoss@smoss_crayola_linux_orl_regress

<Orlando Hardware Regression Results >

Change 85500 on 2003/02/20 by ashishs@fl_ashishs_r400_win

deleting files

Change 85499 on 2003/02/20 by ashishs@fl_ashishs_r400_win

now passes with new VFD change (just removed errorneous data from VFD description) Also removed the shader files since will be generated automatically using the new VFD change.

Change 85495 on 2003/02/20 by smoss@smoss_crayola_linux_orl_regress

changed the queue

Change 85493 on 2003/02/20 by jhoule@jhoule_r400_win_marlboro

Removing addrtile and addrenum: look for them in address.
This reduces the number of libraries to link against, as well as removing static library dependency issues,

Under cmn_lib/src:
- Moved addrenum.{h|c} and addrtile.{h|c} to address directory, removing them for their old location in the process.
- Changed addrtile and addrenum to import/export symbols correctly (using Larry's defines, ready to be used!).
- Changed malloc prototype with an #include <stdlib.h>.

Makefiles:
- Changed to NOT link with the static libraries

Other source files:
- Changed to look at directory 'address' for addrtile.h or addrenum.h

Notes:
- This change was tested with full clobber/sanitize of the environment to make sure old binaries weren't erroneously linked.
- Impact should now be minimal, since programs using the old static libraries were already using the address library.
- Some large modifications might only be tabs->spaces, or trailing spaces removal (changed default editor behavior in the middle of the process to reduce those).

Change 85491 on 2003/02/20 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 85475 on 2003/02/20 by mkelly@fl_mkelly_r400_win_laptop

RTS rectangle walk direction x_dir = 0, y_dir = 1

Change 85419 on 2003/02/19 by csampayo@fl_csampayo_r400

Added r400sq_pressure_context_combo_01 to the list

Change 85377 on 2003/02/19 by smoss@smoss_crayola_linux_orl_regress

lsf test generation

Change 85354 on 2003/02/19 by smoss@smoss_crayola_linux_orl_regress

makefiles for test generation

Change 85334 on 2003/02/19 by ashishs@fl_ashishs_r400_win

updated description in the test

Change 85331 on 2003/02/19 by ashishs@fl_ashishs_r400_win

SQ interpolation test generating 16 shaders each shader exporting different number of paramteres. Thus depending on the number of parameters exported the image gets corresponding number of textures.

Change 85288 on 2003/02/19 by llefebvr@llefebvr_r400_emu_montreal

Adding a context pressure test to the SQ test list. This test exercices the arbiters to make sure no threads pass each other in any circomstance. It has overlaping primitives (32) and uses 4 different shaders of different lenghts. The overlaping primitives are used to make sure the primitives are rendered in the right order.

Change 85275 on 2003/02/19 by kryan@kryan_r400_win_marlboro_XP

- Modify test to use IM_LOAD packet for loading Shader Programs. The default is to use Type-0 PM4 packets,

and this was causing the test to not behave correctly.

- Also had to increase the timeout to 2000 after modifying shader program load method.

Change 85235 on 2003/02/19 by jhoule@jhoule_r400_win_lt

Adding FMT_24_8* formats

Change 85026 on 2003/02/18 by mkelly@fl_mkelly_r400_win_laptop

Real Time Stream primitive type = point, intermixed w/non-RT

Change 84987 on 2003/02/18 by ashishs@fl_ashishs_r400_win

SQ interpolation test generating 16 shaders each shader exporting different number of paramteres

Change 84986 on 2003/02/18 by vbhatia@vbhatia_r400_linux_marlboro

Deleted to save space

Change 84984 on 2003/02/18 by vbhatia@vbhatia_r400_linux_marlboro

Deleted some files to save space

Change 84977 on 2003/02/18 by pmitchel@pmitchel_r400_laptop

moving tp_formatter files to proper place

Change 84974 on 2003/02/18 by georgev@devel_georgev_r400_lin2_marlboro_tott

Changed names.

Change 84948 on 2003/02/18 by jhoule@jhoule_r400_win_lt

Template and common files for Formats4x4 tests

Change 84913 on 2003/02/18 by ashishs@fl_ashishs_r400_win

wrapping each of the 16 parameters (16 textures) on either S, T, or ST (test maybe used for hardware verification)

Change 84899 on 2003/02/18 by mkelly@fl_mkelly_r400_win_laptop

Add CP parameterized tests to Full Emulator Regression and Steve's HW Simulations.

Use script $r400/devel/test_lib/src/chip/gfx/pa_regress/runp to run parameterized tests.

Edit file $r400/devel/test_lib/src/chip/sys/cp/test_list_paramterized for controlling
which tests are run in the full paramterized emulator regression for the non -t option.

Golds for these test cases should be copied to \\fl00fsr02\TestVectors\R400\gold

This checked in file is for full regression running locally in the pa_regress directory which
is one of three options in running the parameterized tests. Just for a reminder, here
are the options to running paramterized tests:

-h display useage
-c use compare list in pa_regress dir by default
-d maximum delta allowed between src and gold image and still PASS
-p copy regression results to Orlando R400 Web
-s get current sync and stamp it on output directory name
-t build and use test_list_parameterized from all unit test_list_parameterized files
-z gzip test output files after compare

1, you can run "runp" script in the unit directory and it will use the local file test_list_paramterized
2, you can run "runp" in the pa_regress directory and it will use the local file test_list_paramterized
3, you can run "runp" in the pa_regress directory with option -t as described above

Change 84890 on 2003/02/18 by mkelly@fl_mkelly_r400_win_laptop

32 single quad prims in one packet, each vert with a different color.

Change 84881 on 2003/02/18 by mkelly@fl_mkelly_r400_win_laptop

...

Change 84553 on 2003/02/14 by ashishs@fl_ashishs_r400_win

updated

Change 84550 on 2003/02/14 by ashishs@fl_ashishs_r400_win

updated

Change 84526 on 2003/02/14 by ashishs@fl_ashishs_r400_win

to test the SQ parameter wrapping 0 and 1 registers for para 0-15 on S and T bits. Loading 16 shaders with each shader export on its respective parameter export.

Change 84510 on 2003/02/14 by jhoule@jhoule_r400_win_marlboro

First try for multiple texture loas with the TM.

Change 84506 on 2003/02/14 by vgoel@fl_vgoel2

added test r400vgt_hos_PNQ_lp_strip_no_proj_01

Change 84505 on 2003/02/14 by mkelly@fl_mkelly_r400_win_laptop

Real Time Stream Line List Primitive

Change 84500 on 2003/02/14 by csampayo@fl_csampayo_r400

Remove test cases that are not complete or obsolete

Change 84486 on 2003/02/14 by pauld@pauld_r400_win_marlboro

sync verification and diag files

Change 84481 on 2003/02/14 by pauld@pauld_r400_win_marlboro

sync verification and diags files

Change 84452 on 2003/02/14 by omesh@omesh_r400_linux_marlboro_only_devel

Changed tests to use the new COLOR_SURFACE class instead of the PIXEL_SURFACE class.
Dumps using the render state for the PIXEL_SURFACE class are not implemented.

Change 84441 on 2003/02/14 by mkelly@fl_mkelly_r400_win_laptop

Change base color from C1 to C31

Change 84406 on 2003/02/14 by mkelly@fl_mkelly_r400_win_laptop

32 1 quad prims, random color on each vert, each prim

Change 84403 on 2003/02/14 by csampayo@fl_csampayo2_r400

Add new tests created by George

Change 84333 on 2003/02/13 by ygiang@ygiang_r400_pv2_marlboro

added: mova random vtx test case

Change 84293 on 2003/02/13 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint

Change 84290 on 2003/02/13 by mkelly@fl_mkelly_r400_win_laptop

RTS intertwined with Hi-Z viz query

Change 84286 on 2003/02/13 by ashishs@fl_ashishs_r400_win

SQ 16 texture interpolation test (changed cl_barycentric_perspective tests with 16 textures)

Change 84281 on 2003/02/13 by omesh@omesh_r400_linux_marlboro_only_devel

Used the tile surface class instead of the earlier one and verified that the test dumps a tile buffer. However, this
probably doesn't contain accurate data as the emulator doesn't do tile buffer operations. When I run this on h/w, it will
hopefully produce a valid tile buffer dump which can be used by Kevin's code and DumpView for an intelligent compare. Will
try that next....

Change 84276 on 2003/02/13 by omesh@omesh_r400_linux_marlboro_only_devel

Added another testcase for Paul Vella (which seems to assert on the
emulator for no aparent reason). I have contacted Larry about this.

Change 84239 on 2003/02/13 by ashishs@fl_ashishs_r400_win

to test sq interpolation with 16 textures

Change 84235 on 2003/02/13 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 84202 on 2003/02/13 by mkelly@fl_mkelly_r400_win_laptop

Finalized using RB_TILECONTROL to mimic Hi-Z culling by returning zeroed masks

Change 84174 on 2003/02/13 by ashishs@fl_ashishs_r400_win

clipping with wrapping texture in S and T for 16 parameters

Change 84167 on 2003/02/13 by georgev@devel_georgev_r400_lin2_marlboro_tott

Added random test.

Change 84162 on 2003/02/13 by ashishs@fl_ashishs_r400_win

to test the SQ wrapping 0 and 1 registers for para 0 -15 on S and T bits.

Change 84118 on 2003/02/13 by mkelly@fl_mkelly_r400_win_laptop

Disable poly offset for non-RT

Change 84112 on 2003/02/13 by mkelly@fl_mkelly_r400_win_laptop

Remove some comments in _03
new VIZ Query pixel post hi-z kill test
new VIZ Query pixel post detail mask kill test

Change 84110 on 2003/02/13 by mkelly@fl_mkelly_r400_win_laptop

update rb settings...

Change 84106 on 2003/02/13 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 84018 on 2003/02/12 by ashishs@fl_ashishs_r400_win

changed colors for better visibility

Change 84014 on 2003/02/12 by omesh@omesh_r400_linux_marlboro_only_devel

Added a special testcase that Paul Vella wanted to check out.

Change 83987 on 2003/02/12 by ashishs@fl_ashishs_r400_win

rearranged texture coordinates

Change 83978 on 2003/02/12 by omesh@omesh_r400_linux_marlboro_only_devel

Added all tile buffer base address programming. This test should now have tiling enabled. Will work with Kevin to make sure it produces the
right data for compressed multisampled images to display correctly.

Change 83921 on 2003/02/12 by omesh@omesh_r400_linux_marlboro_only_devel

Added some more testcases and also fixed some.

Change 83880 on 2003/02/12 by ashishs@fl_ashishs_r400_win

enabled clipping

Change 83879 on 2003/02/12 by ashishs@fl_ashishs_r400_win

removed extra data in VTX

Change 83876 on 2003/02/12 by georgev@devel_georgev_r400_lin2_marlboro_tott

Change for windows disk.

Change 83870 on 2003/02/12 by ashishs@fl_ashishs_r400_win

clipping 16 textured triangle_list

Change 83834 on 2003/02/12 by georgev@devel_georgev_r400_lin2_marlboro_tott

Made useful.

Change 83832 on 2003/02/12 by georgev@devel_georgevhw_r400_win_marlboro

no comment

Change 83830 on 2003/02/12 by georgev@ma_georgev

Temp file for messing about.

Change 83828 on 2003/02/12 by ashishs@fl_ashishs_r400_win

Test to check context switching in SQ using large number of paramters in first packet and very less parameters in second packet and switching between the 2 types

The test is similar to the r400sq_1col_15tex_interp_01 except that this test
generates 32 packets. The first 16 packets have alternating data as polygon with
15 textures and gouraud shaded triangle. The next 16 packets are randomly
switched between the 2 types of packets.

Change 83815 on 2003/02/12 by georgev@devel_georgevhw_r400_win_marlboro

Took out sleep function so it would compile and run on windows.

Change 83812 on 2003/02/12 by ashishs@fl_ashishs_r400_win

test updated to generate shaders using VFD and deleting the old shaders

Change 83782 on 2003/02/12 by ashishs@fl_ashishs_r400_win

made gouraud triangle hit only 4 pixels (single quad)

Change 83779 on 2003/02/12 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 83766 on 2003/02/12 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint

Change 83761 on 2003/02/12 by mkelly@fl_mkelly_r400_win_laptop

Add full tracking and reporting of the SQ block.

Change 83731 on 2003/02/11 by ashishs@fl_ashishs_r400_win

adding test list for SQ tests

Change 83730 on 2003/02/11 by ashishs@fl_ashishs_r400_win

vtx and pix shaders for r400sq_1col_15tex_interp_01

Change 83727 on 2003/02/11 by ashishs@fl_ashishs_r400_win

Test to check context switching in SQ using large number of paramters in first packet and very less parameters in second packet.

Change 83666 on 2003/02/11 by ygiang@ygiang_r400_win_marlboro_p4

fixed:pixel vector sizes

Change 83642 on 2003/02/11 by mkelly@fl_mkelly_r400_win_laptop

Handle only blending of textures in pixel shader. Requires setting
CONSTANT 1 at the test level for the first blend stage.
See r400sc_tri_16_par_64_dwords_02.cpp for example useage.

Change 83640 on 2003/02/11 by mkelly@fl_mkelly_r400_win_laptop

Update gold

Change 83630 on 2003/02/11 by kryan@kryan_r400_win_marlboro

- Modify test to more closely match r400rb_multisample_fragmented*.cpp tests

by setting the number of samples in the RB_COLOR0_INFO and RB_DEPTH_INFO

registers.

- Modify test to pass RENDER_STATE into Dump() function of COLOR_SURFACE
so

that proper header is generated for DumpView.

Change 83617 on 2003/02/11 by ashishs@fl_ashishs_r400_win

updated

Change 83583 on 2003/02/11 by georgev@devel_georgev_r400_lin2_marlboro_tott

Added multiple context tests.

Change 83577 on 2003/02/11 by omesh@omesh_r400_linux_marlboro_only_devel

Even though there are some assertions I get on the emulator for some of the testcases, I
believe
these assertions are invalid. I will clarify this with Larry.

Change 83559 on 2003/02/11 by ashishs@fl_ashishs_r400_win

changed test name

Change 83551 on 2003/02/11 by mkelly@fl_mkelly_r400_win_laptop

update RB setup

Change 83526 on 2003/02/11 by mkelly@fl_mkelly_r400_win_laptop

update rb setup

Change 83519 on 2003/02/11 by mkelly@fl_mkelly_r400_win_laptop

update rb setup

Change 83511 on 2003/02/11 by mkelly@fl_mkelly_r400_win_laptop

Update RB setup

Change 83496 on 2003/02/11 by mkelly@fl_mkelly_r400_win_laptop

Update RB setup for MSAA

Change 83490 on 2003/02/11 by mkelly@fl_mkelly_r400_win_laptop

Update RB setup for MSAA

Change 83489 on 2003/02/11 by mkelly@fl_mkelly_r400_win_laptop

Disable polyoffset in nonRTS, disable stipple in RTS

Change 83487 on 2003/02/11 by jhoule@jhoule_r400_win_marlboro

Addind the FMT_*_EXPAND formats

Change 83483 on 2003/02/11 by ashishs@fl_ashishs_r400_win

to test SQ parameter wrapping for 8 textures with clipping(initial checkin)

Change 83479 on 2003/02/11 by mkelly@fl_mkelly_r400_win_laptop

Disable polyoffset in nonRTS packets.

Change 83476 on 2003/02/11 by mkelly@fl_mkelly_r400_win_laptop

Disable polyoffset in the NON RTS packets.

Change 83423 on 2003/02/10 by ygiang@ygiang_r400_win_marlboro_p4

added: more mova tests

Change 83406 on 2003/02/10 by ygiang@ygiang_r400_win_marlboro_p4

added: another random mova test for sq

Change 83335 on 2003/02/10 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 83266 on 2003/02/10 by mkelly@fl_mkelly_r400_win_laptop

Add new dumps...

Change 83233 on 2003/02/10 by mkelly@fl_mkelly_r400_win_laptop

Use all 4 SQ parameters for RT streams...

Change 83218 on 2003/02/10 by jhoule@jhoule_r400_win_marlboro

Added a bunch of useful textures; closer to regression suite integration.

Change 82977 on 2003/02/07 by markf@markf_r400_lt_marlboro

Modified r400rb_zwave tests to use the tile buffer (depth will now be compressed).

Change 82948 on 2003/02/07 by vromaker@vromaker_r400_linux_marlboro

one pixel vector version

Change 82945 on 2003/02/07 by ygiang@ygiang_r400_win_marlboro_p4

add: new sp const opcode tests

Change 82937 on 2003/02/07 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 82921 on 2003/02/07 by csampayo@fl_csampayo_r400

Add back some missing tests

Change 82912 on 2003/02/07 by smoss@smoss_crayola_linux_orl_regress

changed unix output directory and hopefully output from stderr

Change 82907 on 2003/02/07 by ygiang@ygiang_r400_win_marlboro_p4

added: sp dot2add test with swizzle

Change 82884 on 2003/02/07 by kryan@kryan_r400_win_marlboro

Add PrimLib support for multisample Color Surfaces.

Still in progress.

COLOR_SURFACE

- Modified Dump(RENDER_STATE&) function to reformat the

dump for multisample Color Surfaces so that DumpView can

display them correctly.

PIXEL_SURFACE

- Modified Fill_Solid() function to take into account

the number of samples for multisample Color Surfaces.

Please see gfx/rb/r400rb_msaa.cpp

r400rb_basic_multisample.cpp

r499rb_multisample_fragmented.cpp

for examples of how to setup your COLOR_SURFACE class for

multisample Color Surfaces.

gfx/rb/tests

- Modify to pass RENDER_STATE into COLOR_SURFACE::Dump() routine.

r400_multisample_fragmented_resolve.cpp

- Make sure to dump multisample Color Surface before modifying

RENDER_STATE for resolve Color Surface.

Change 82816 on 2003/02/07 by mkelly@fl_mkelly_r400_win_laptop

golds for Bob...

Change 82814 on 2003/02/07 by viviana@viviana_crayola_linux_orl

Removed the write/read of PA_SC_ENHANCE and PA_SC_WINDOW_OFFSET since they are read by the SC.

Change 82808 on 2003/02/07 by mkelly@fl_mkelly_r400_win_laptop

Validate dummy quad deallocation in pixel vector buffer is good.

Change 82792 on 2003/02/07 by viviana@viviana_crayola_linux_orl

Removed PA_SC_CNTL_STATUS from both tests (read only register). Changed the data written to the PA_SC_FIFO_SIZE so that the maximum of 84 would not be exceeded for the PRIM FIFO.

Change 82705 on 2003/02/06 by csampayo@fl_csampayo_r400

Updated to include first known tests

Change 82558 on 2003/02/06 by mkelly@fl_mkelly_r400_win_laptop

Comment out poly_offset_03/10 until debugged

Change 82518 on 2003/02/06 by jhoule@jhoule_r400_win_marlboro

Added Fake_SP_TP_Formatter::prepare function to set things up:
- Put old hardware-accurate code in there
- Replaced NaNs with 0, since this is what the HW does (that zeroing is done in the FSTF)

Changed looping code to better match the xyzw_parity.

TPBlender:
- Added kCS and kCE to localize loop count boundary calculations
- Added mOutputPtr to speed up accesses using pixel and channel IDs
- Changed exponent adjust code to work only on modified channels

Change 82516 on 2003/02/06 by mkelly@fl_mkelly_r400_win_laptop

Reset line stipple at each primitive

Change 82507 on 2003/02/06 by mkelly@fl_mkelly_r400_win_laptop

Initialize stipple state = 0x0

Change 82504 on 2003/02/06 by mkelly@fl_mkelly_r400_win_laptop

Initialize pa_sc_line_stipple_state = 0x0

Change 82503 on 2003/02/06 by markf@markf_r400_linux_marlboro

Random pixel shader test

Change 82494 on 2003/02/06 by mkelly@fl_mkelly_r400_win_laptop

Enter some sq sx tests into main test_list_paramterized

Change 82490 on 2003/02/06 by mkelly@fl_mkelly_r400_win_laptop

Enable SX and SQ for regression and gold compares using runp

Change 82474 on 2003/02/06 by mkelly@fl_mkelly_r400_win_laptop

Parameterized cases for regressions and gold candidates

Change 82444 on 2003/02/06 by mkelly@fl_mkelly_r400_win_laptop

Update for debug

Change 82441 on 2003/02/06 by llefebvr@llefebvr_r400_emu_montreal

not requesting enough GPRs...

Change 82438 on 2003/02/06 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 82433 on 2003/02/06 by mkelly@fl_mkelly_r400_win_laptop

Fix interpolator selection on shading...

Change 82431 on 2003/02/06 by mkelly@fl_mkelly_r400_win_laptop

Fix interpolator selection on shading.

Change 82429 on 2003/02/06 by mkelly@fl_mkelly_r400_win_laptop

Simplified version of r400sc_rts_12 for regress_e

Change 82423 on 2003/02/06 by ashishs@fl_ashishs_r400_win

converted r400cl_ucp_combos_01 to write into the CL_ENHANCE (num of Clip Seq and Clip Prim Seq Stall) register and monitor any changes. Also read the CL_BUSY register

Change 82420 on 2003/02/06 by kevino@kevino_r400_win_marlboro

Added 512x512 texture point sampled to 512x512 square case

Change 82396 on 2003/02/06 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 82392 on 2003/02/06 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 82391 on 2003/02/06 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 82386 on 2003/02/06 by mkelly@fl_mkelly_r400_win_laptop

Fix stipple auto reset cntl, lines are now clipped, interpolate p0, not p1

Change 82378 on 2003/02/06 by mkelly@fl_mkelly_r400_win_laptop

Stipple auto reset control = true always now

Change 82230 on 2003/02/05 by mkelly@fl_mkelly_r400_win_laptop

Shortened to assertion.

Change 82220 on 2003/02/05 by mkelly@fl_mkelly_r400_win_laptop

Fix stipple auto_reset_cntl, enable CL, change line generation method

Change 82191 on 2003/02/05 by csampayo@fl_csampayo_r400

Forgot to save changes for this one

Change 82184 on 2003/02/05 by csampayo@fl_csampayo_r400

Increase max memory size for tests. Update spreadsheet

Change 82178 on 2003/02/05 by ashishs@fl_ashishs_r400_win

tests with various primitive types and edgeflag and ucp clipping. removing textures from the previous tests and using gouraud shading to check the interpolation

Change 82172 on 2003/02/05 by ygiang@ygiang_r400_win_marlboro_p4

fixed: z16 does not support 32x32 depth buffer

Change 82090 on 2003/02/05 by mkelly@fl_mkelly_r400_win_laptop

Fix syntax error introduced in last checkin...

Change 82083 on 2003/02/05 by mkelly@fl_mkelly_r400_win_laptop

Fix syntax error from previous checkin.

Change 82082 on 2003/02/05 by mkelly@fl_mkelly_r400_win_laptop

Fix syntax error from previous check in.

Change 82081 on 2003/02/05 by mkelly@fl_mkelly_r400_win_laptop

Fixed syntax error preventing build from previous check in by Omesh.

Change 82067 on 2003/02/05 by mkelly@fl_mkelly_r400_win_laptop

Check point...

Change 82043 on 2003/02/05 by llefebvr@llefebvr_r400_emu_montreal

This is a test problem. R0 is used whenever the pixel shader is sending front face information. Thus it is parameter 0 which should be FLAT shaded NOT parameter 1 (parameter 1 is the generated one with XYST). I changed line 669 in the test from FFFD to FFFE (flat shade param 0 instead of 1). Now the test works great.

Change 82035 on 2003/02/05 by mkelly@fl_mkelly_r400_win_laptop

Flat/Gouraud Shade interpolation bug

Change 81927 on 2003/02/04 by ashishs@fl_ashishs_r400_win

combination of rectangular points clipped by 6 clipping planes

Change 81889 on 2003/02/04 by omesh@omesh_r400_linux_marlboro_only_devel

Added RB programming related to multisampling so that the RB emulator does not assert when those changes are put in by Larry. I ran regress_e and the
tests seem to work.

Change 81877 on 2003/02/04 by mkelly@fl_mkelly_r400_win_laptop

Fix VFD to remove invalid texture fetches which became a problem after
check-in #81131.

Fix VFD for vtx kill, now fetches x channel instead of z during vfetch.

Change 81853 on 2003/02/04 by ygiang@ygiang_r400_win_marlboro_p4

added: rand addressing for mova test

Change 81843 on 2003/02/04 by ygiang@ygiang_r400_pv_marlboro

added: mova tests file

Change 81798 on 2003/02/04 by ashishs@fl_ashishs_r400_win

The goal of the test is to check rectangular points. This test has 64 points with varying width and height of the point so as to get rectangular points BY setting point size in vtx data and clipping enabled,so that the CL registers for point size are used.
The test shows that rectangular points is not possible when the point size is
set in the vertex data. Even putting unequal ratios in the X RADIUS and Y RADIUS of
CL point size registers, the CL just uses X RADIUS to construct points when the point
size is set thru the vertex data.

Change 81791 on 2003/02/04 by mkelly@fl_mkelly_r400_win_laptop

RTS intertwined with Viz Query and kill_pix_post_detail_mask.

Change 81785 on 2003/02/04 by mkelly@fl_mkelly_r400_win_laptop

RTS intertwined with VIZ_QUERY and SC Kill pix post hi z...

Change 81769 on 2003/02/04 by smoss@smoss_crayola_win

increased timeout

Change 81731 on 2003/02/04 by mkelly@fl_mkelly_r400_win_laptop

Viz Query intertwined with RT streams, complete...

Change 81672 on 2003/02/03 by ygiang@ygiang_r400_win_marlboro_p4

fixed: pixel kill tests

Change 81632 on 2003/02/03 by ygiang@ygiang_r400_win_marlboro_p4

fixed: test for sp

Change 81598 on 2003/02/03 by ygiang@ygiang_r400_win_marlboro_p4

added: more pixel kill tests

Change 81590 on 2003/02/03 by llefebvr@llefebvr_r400_emu_montreal

Interpolation tests.

Change 81553 on 2003/02/03 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint on expanding test...

Change 81541 on 2003/02/03 by ygiang@ygiang_r400_win_marlboro_p4

added: pixel kill test

Change 81484 on 2003/02/03 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint, 2 viz querys / 1 RTS

Change 81453 on 2003/02/03 by llefebvr@llefebvr_r400_emu_montreal

fixing the segmentation fault on Linux.

Change 81420 on 2003/02/03 by ctaylor@ctaylor_crayola_linux_orl

Added PA_SU_POINT_MINMAX description at request of driver.
Changed usage of cmdKeyObj in r400sc_rand.cpp to compile

Change 81412 on 2003/02/03 by ashishs@fl_ashishs_r400_win

updated

Change 81390 on 2003/02/03 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint..

Change 81373 on 2003/02/03 by mkelly@fl_mkelly_r400_win_laptop

Create vertex buffer to view for debugging

Change 81180 on 2003/01/31 by jhoule@jhoule_r400_win_marlboro

Updated flavor script.
Updated format files.

Change 81178 on 2003/01/31 by jhoule@jhoule_r400_win_marlboro

New simple formats 4x4 tests

Change 81164 on 2003/01/31 by georgev@devel_georgevh2_r400_win_marlboro

Fixed bad shader. Missing a line.

Change 81145 on 2003/01/31 by ashishs@fl_ashishs_r400_win

rectangular points using point size in SU registers

Change 81140 on 2003/01/31 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint Viz Query with RTS

Change 81134 on 2003/01/31 by omesh@omesh_r400_linux_marlboro_only_devel

Found the problem with Primlib and in this test, I have worked around that problem by not using 2 render states, and also
not using the equal to (=) assignment operator, as it destroys some data within the source operand render state. Will
file a seperate bug under Primlib for this symptom.

Change 81122 on 2003/01/31 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint on Viz Query with RTS...

Change 81116 on 2003/01/31 by jhoule@jhoule_r400_win_marlboro

Update to the Fake_SP_TP_Formatter (or FSTF, for short).

- Added structs to the FSTF: allows runtime values to be managed.
- Split parsing code from conversion code.
- Moved swizzles and NaNs from TexturePipe to TPBlender.
- Removed mIsDegamma in TPBlender as this is no longer needed (formats are split with _AS_4x16 flavors)

Left to do:
- Do rfExpand outside of convert_hw (removes need for mSkipRFExpand variable)
- Fix mantissa clamping issue for 16.16 or 32.0 float conversion (emulator currently rounds)
- Verify that the %2 parity works for channels masks

Change 80962 on 2003/01/30 by ygiang@ygiang_r400_pv_marlboro

fixed: sp mova tests

Change 80918 on 2003/01/30 by llefebvr@llefebvre_laptop_r400_emu

Adding memory export tests to nightly RBRC regression.
Adding 2 more mova tests to stress test the SQ.

Change 80853 on 2003/01/30 by mkelly@fl_mkelly_r400_win_laptop

Better clamping for greater HW range testing inside of HW boundary

Change 80839 on 2003/01/30 by mkelly@fl_mkelly_r400_win_laptop

Enable guard band clipping to avoid exceeding hw boundary with lines
Fix bug with indice generation

Change 80817 on 2003/01/30 by ashishs@fl_ashishs_r400_win

corrected ucp_ps_modes

Change 80809 on 2003/01/30 by omesh@omesh_r400_linux_marlboro_only_devel

Another modification of the test in which I use the same render state object instead of 2, but I add the 2D Shadow Register
programming after I "add" the 3D register programming. Although this seems to work better than when I used 2 seperate
render state objects (I do get some primitives drawn on the screen), the framebuffer is still missing most of the
primitives. I will file this comment in addition to the already existing bug.

Change 80799 on 2003/01/30 by ashishs@fl_ashishs_r400_win

to test SU point size set in state registers with different ps_ucp_modes. The test has a generalised test structure where the point size could be toggled between the vertex xport and the SU state registers keeping the output same.

Change 80760 on 2003/01/30 by omesh@omesh_r400_linux_marlboro_only_devel

Checked in trial of code that uses an isolated, seperate render_state object for setting 2D shadow registers,
after inheriting an entire render state (including vertex buffer parameters) from another render state object.
So far, it still doesn't seem to work, so I will redirect Bugzilla ID 1189 to Kevin Ryan for possible Primlib
bugs.

Change 80739 on 2003/01/30 by ashishs@fl_ashishs_r400_win

to test vertex xport point size thru shaders with different ps_ucp_modes. The test has a generalised test structure where the point size could be toggled between the vertex xport and the SU state registers keeping the output same.

Change 80707 on 2003/01/30 by mkelly@fl_mkelly_r400_win_laptop

Update comments, RTS with SC quad order enable toggling...

Change 80705 on 2003/01/30 by jhoule@jhoule_r400_win_marlboro

Adding templates to comment a bit what input format the fake_sp_tp_formatter takes.

Change 80686 on 2003/01/30 by mkelly@fl_mkelly_r400_win_laptop

RTS and SC FIFO sizing combinations...

Change 80679 on 2003/01/30 by mkelly@fl_mkelly_r400_win_laptop

RTS combinations with Vtx and Pix pipes 0/2 disabled with
SC one quad per clock toggled, shader back pressure,
interpolator shading toggling

Change 80514 on 2003/01/29 by jhoule@jhoule_r400_win_marlboro

Better SP_TP_Formatter testbench support.
- Renamed TP_SP_Formatter to SP_TP_Formatter everywhere
- Added rfExpand skipping control in the TPBlender
- Added readTTs/writeTTs methods to help parsing of the formatter

Bug fixes:

Fixed 16b signed fixed point conversion (was using a bias of 2^14 instead of 2^15).

Fixed gradient precision loss.
Uses grad.putField(grad.getField()) instead of grad.floor() assignment which only works in _hw mode.

Changed 16b float GradientType from 15 to 17 as third parameter.

Change 80487 on 2003/01/29 by csampayo@fl_csampayo_r400

Add missing write to debug reg cntl select field

Change 80479 on 2003/01/29 by mkelly@fl_mkelly_r400_win_laptop

Vtx and pix pipes 2 and 3 disabled with RTS triangles and rectangles and non-RTS stipple lines, complete

Change 80476 on 2003/01/29 by markf@markf_r400_linux_marlboro

Adding basic test for ARRAY_3D_SLICE

Change 80442 on 2003/01/29 by omesh@omesh_r400_linux_marlboro_only_devel

Who let the dogs out? (Someone clobbered an old change in the test which made it run for too long on the all and random testcases). I
restored my old change.

Change 80398 on 2003/01/29 by csampayo@fl_csampayo_r400

Adding new VGT test for missing reg coverage

Change 80305 on 2003/01/29 by mkelly@fl_mkelly_r400_win_laptop

Check stippled line integrity with real time streams, complete.

Change 80299 on 2003/01/29 by vromaker@vromaker_r400_linux_marlboro

one pixel vector version

Change 80296 on 2003/01/29 by markf@markf_r400_linux_marlboro

Fixed background color for sub_rect tests

Change 80271 on 2003/01/29 by mkelly@fl_mkelly_r400_win_laptop

Update 11, checkpoint on 12

Change 80061 on 2003/01/28 by ashishs@fl_ashishs_r400_win

The test has 110 cases. The value of the field "VTX_KILL" in the CLIP_CNTL register
is contsantly toggled randomly as well as in a sequence to generate "OR" or "AND" modes. Also the vtx kill flags in the vtx data are toggled randomly and in sequence to generate 110 cases such that all different combinations of vertex kill flag with a single triangle list are covered.

Change 80054 on 2003/01/28 by omesh@omesh_r400_linux_marlboro_only_devel

Restored fragment color parameters. Later will add more corner cases.

Change 80051 on 2003/01/28 by omesh@omesh_r400_linux_marlboro_only_devel

Added 2 register fields programming for future expansion for R450.
Also added some missing programming in some tests.
These source files are a bit messy with comments and dead code. Will
clean it up later....

Change 80027 on 2003/01/28 by jayw@jayw_r400_linux_marlboro

new regression scripts.

Change 79999 on 2003/01/28 by markf@markf_r400_linux_marlboro

Added resolve test cases

Change 79984 on 2003/01/28 by mkelly@fl_mkelly_r400_win_laptop

Polymode RTS test

Change 79923 on 2003/01/28 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 79914 on 2003/01/28 by grayc@chip_regress_orl

changes for new arch

Change 79911 on 2003/01/28 by ygiang@ygiang_r400_pv_marlboro

more mova test for debug

Change 79886 on 2003/01/28 by grayc@chip_regress_orl

added variables

Change 79829 on 2003/01/27 by llefebvr@llefebvre_laptop_r400_emu

Fixing the access violation exeption.
Also making RealDOT the default setting for the emulator. Emulator should now be 100% HW accurate in the SP.

Change 79826 on 2003/01/27 by ygiang@ygiang_r400_pv_marlboro

added: more mova tests for debug

Change 79818 on 2003/01/27 by csampayo@fl_csampayo_r400

Adding new VGT fifo tests

Change 79784 on 2003/01/27 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint..

Change 79715 on 2003/01/27 by georgev@devel_georgev_r400_lin2_marlboro

Extra tests for Vic.

Change 79619 on 2003/01/27 by ashishs@fl_ashishs_r400_win

updated

Change 79613 on 2003/01/27 by ashishs@fl_ashishs_r400_win

added test description and updated tracker

Change 79610 on 2003/01/27 by kmahler@kmahler_r400_win_devel_views

Added "random_pixel_shader" test case code to existing tests to generate a
Random Pixel Shader.

This was done to test the generator and to give people example code of how to
create a random pixel shader.

"random_centers_and_centroids_state_switching_01" test was taken from

"chip/gfx/sc/r400sc_centers_and_centroids_state_switching_01".

Change 79593 on 2003/01/27 by mkelly@fl_mkelly_r400_win_laptop

Test bug fix, all verts now have 12 parameters.

Change 79489 on 2003/01/25 by markf@markf_r400_linux_marlboro

Fixed numer of samples

Change 79487 on 2003/01/25 by markf@markf_r400_linux_marlboro

random multi-sample test

Change 79419 on 2003/01/24 by jayw@jayw_r400_linux_marlboro

added tb to regress5

Change 79415 on 2003/01/24 by jhoule@jhoule_r400_win_marlboro

Dedicated sp_tp_formatter testbench

Added fake TP_SP_Formatter.

TexturePipe and other classes now get exported under Windows in order to instanciate it outside of full chip setups. Some weird issues were encountered (symbols wouldn't get exported); hopefully, this will work for everyone.

UNFINISHED

Change 79393 on 2003/01/24 by ashishs@fl_ashishs_r400_win

to test the vertex reordering of the CL

Change 79308 on 2003/01/24 by markf@markf_r400_linux_marlboro

Fixed sub_* tests

Change 79288 on 2003/01/24 by jhoule@jhoule_r400_win_marlboro

Some regression tests

Change 79252 on 2003/01/24 by ashishs@fl_ashishs_r400_win

varying gbands with valid data as well as nan and inf's to get 100% register coverage

Change 79235 on 2003/01/24 by mkelly@fl_mkelly_r400_win_laptop

RT provoking vertex looking good through interpolator on one parameter.

Change 79207 on 2003/01/24 by mkelly@fl_mkelly_r400_win_laptop

Only compare <test_name>.rd_r during full emu regression on reg read tests

Change 79193 on 2003/01/24 by smoss@smoss_crayola_linux_orl_regress

reverting changes

Change 79174 on 2003/01/24 by mkelly@fl_mkelly_r400_win_laptop

Add modified and shortened version of r400sc_rts_09 (back face check on nonRT vs RT prims) to regress_e

Change 79162 on 2003/01/24 by mkelly@fl_mkelly_r400_win_laptop

Update comments in shader

Change 79161 on 2003/01/24 by mkelly@fl_mkelly_r400_win_laptop

Final, validating Pixel Shader face bit detection from sc_sp with nonRT and RT primtives

Change 79148 on 2003/01/24 by omesh@omesh_r400_linux_marlboro_only_devel

Created a seperate version of the ROP3 tests (renamed) using the 2D
Shadow Registers. Jay mentioned that we need both versions, so I
seperated them.

Change 79147 on 2003/01/24 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 79137 on 2003/01/24 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint, RTS face bit, detect in Pixel Shader...

Change 79105 on 2003/01/23 by jayw@jayw_r400_linux_marlboro

2D shadow register support almost complete.

Change 79064 on 2003/01/23 by ygiang@ygiang_r400_pv2_marlboro

indirect buffer for constants in test

Change 79022 on 2003/01/23 by ygiang@ygiang_r400_pv2_marlboro

added: indirect buffer to test

Change 79020 on 2003/01/23 by jayw@jayw_r400_linux_marlboro

jay's regression scripts

Change 79018 on 2003/01/23 by omesh@omesh_r400_linux_marlboro_only_devel

Eliminated even more 2 testcases per test of 16 bit "16_number_float"
which are no longer valid.

Change 78986 on 2003/01/23 by omesh@omesh_r400_linux_marlboro_only_devel

Removed all occurences of the color_16_number_float testcases which are
no longer valid. Instead color_16_float_number_float is now the valid
testcase of using 16 bit floating point in RB.

Change 78960 on 2003/01/23 by omesh@omesh_r400_linux_marlboro_only_devel

Fixed tests so they produce more visually predictive results.
Later, will split the larger testcases so they don't get rejected by
LSF.

Change 78876 on 2003/01/23 by mkelly@fl_mkelly_r400_win_laptop

Validate face bit in pixel shader for multi-tile coverage prims

Change 78870 on 2003/01/23 by omesh@omesh_r400_linux_marlboro_only_devel

Added a "dest_alpha_problem" testcase to recreate the problem that Dan Willhite from s/w filed a bug for
(Blue Source triangle blended with Red Destination triangle using the ADD function), and it seems to

produce the right result on the verification environment (Magenta triangle).

Change 78867 on 2003/01/23 by ashishs@fl_ashishs_r400_win

toggling the xy_nan_retain, z_nan_retain and w_nan_retain randomly from PA_CL_CLIP_CNTL register

Change 78855 on 2003/01/23 by csampayo@fl_csampayo2_r400

Removed EID# 21 since, causing RB to write garbage to screen

Change 78851 on 2003/01/23 by omesh@omesh_r400_linux_marlboro_only_devel

Added explicit Color Compare Mask to enable all compares. Restored size
of triangles and placement on screen. The EQ and NEQ seems to work for
most cases but not all. I will further investigate.

Change 78823 on 2003/01/23 by omesh@omesh_r400_linux_marlboro_only_devel

Made triangles larger and fewer triangles per row, for easier debug, for
now.

Change 78797 on 2003/01/23 by smoss@smoss_crayola_linux_orl_regress

hopefully it is right now

Change 78741 on 2003/01/22 by smoss@smoss_crayola_linux_orl_emu_regress

update

Change 78729 on 2003/01/22 by smoss@smoss_crayola_win

change default directory for linux:

Change 78608 on 2003/01/22 by georgev@devel_georgev_r400_lin2_marlboro

Lowered long loop counts to smaller values.

Change 78559 on 2003/01/22 by mkelly@fl_mkelly_r400_win_laptop

Final fixes to test...

Change 78538 on 2003/01/22 by mkelly@fl_mkelly_r400_win_laptop

Move wait_rt_idle to immediately after rt stream reg writes.

Change 78466 on 2003/01/22 by llefebvr@llefebvr_r400_emu_montreal

More memory export tests and bug fixes in the SX (wrapping problem in the export buffer for large quantity of exports).

Change 78447 on 2003/01/22 by mkelly@fl_mkelly_r400_win_laptop

Changed wait_rt_idle to occur before wait_gfx_idle.

Change 78430 on 2003/01/22 by mkelly@fl_mkelly_r400_win_laptop

Temporarily isolate problem in test

Change 78418 on 2003/01/22 by mkelly@fl_mkelly_r400_win_laptop

Change shader names to test name and add missing simple shader...

Change 78415 on 2003/01/22 by mkelly@fl_mkelly_r400_win_laptop

Add triangle prim type real time stream...

Change 78345 on 2003/01/21 by ashishs@fl_ashishs_r400_win

updated

Change 78261 on 2003/01/21 by omesh@omesh_r400_linux_marlboro_only_devel

Changed test to use 2D registers (in a new render state, added later)
instead of the 3D registers related to ROP3. Verified that the test
compiles and runs, but haven't verified that it produces the same result
as earlier on the emulator or h/w.

Change 78240 on 2003/01/21 by ygiang@ygiang_r400_pv2_marlboro

added: sq mova test for debug

Change 78143 on 2003/01/21 by llefebvr@llefebvr_r400_emu_montreal

New memory export tests that do 2 blocks of export. I striped down Vineet's test so that they would run faster on the simulator.

Change 78091 on 2003/01/20 by ashishs@fl_ashishs_r400_win

initial checkin

Change 78076 on 2003/01/20 by georgev@devel_georgev_r400_lin2_marlboro

Made eq test triangle smaller.

Change 78061 on 2003/01/20 by kryan@kryan_r400_linux_marlboro

Fix error in Makefile that causes link error on Linux. Commented out new code to
explicitly link in addr* libraries on Unix until error is resolved.

Change 78054 on 2003/01/20 by kryan@kryan_r400_win_marlboro

Modified Makefile since r400rb_tb.cpp needs to use address and addrenum libraries

on Windows.

Previously this test was not linking correctly on Windows since it

calls a function from the addrenum library which was not being linked in on

Windows.

Change 78050 on 2003/01/20 by markf@markf_r400_win_marlboro

Reset fast_clear_enables only if doing a subsequent clear or expand.

Change 77933 on 2003/01/20 by markf@markf_r400_win_marlboro

Updated so it doesn't send extra state at the end of the test

Change 77921 on 2003/01/20 by omesh@omesh_r400_linux_marlboro_only_devel

Added tests for chroma keying. These are fairly extensive for source
kill, although I still need to add some more select functions. I also
need to add random testcases.
These display some emulator bugs, which I am going to enter on Bugzilla.

Change 77918 on 2003/01/20 by smoss@smoss_crayola_linux_orl_emu_regress

added rts files

Change 77917 on 2003/01/20 by ashishs@fl_ashishs_r400_win

permuting edgeflags with different primitve types with LINE FILL , and guard band
clipping

Change 77889 on 2003/01/20 by georgev@devel_georgev_r400_lin2_marlboro

Changed JMP to CALL because that's what's being tested. Now the test works.

Change 77857 on 2003/01/20 by ashishs@fl_ashishs_r400_win

permuting edgeflags for different primitve types with PointFill

Change 77742 on 2003/01/18 by markf@markf_r400_linux_marlboro

Defaulted the Z component of the texture coordinate to 0.0

Change 77709 on 2003/01/18 by markf@markf_r400_linux_marlboro

Disabled blending on the rb_color_formats tests when testsing 16bit integer formats.

Change 77595 on 2003/01/17 by ygiang@ygiang_r400_pv_marlboro

fixed: alu constants needed for tests.

Change 77582 on 2003/01/17 by omesh@omesh_r400_linux_marlboro_only_devel

Changed device address passed to the ZPASS_ADDR field, as it seems to be missing 2
bits, I right
shift the LSBs to lose the 2 bits, but the emulator still doesn't seem to write the correct
result
to memory.

Change 77554 on 2003/01/17 by mkelly@fl_mkelly_r400_win_laptop

Rectangle and triangle real time stream initial functional

Change 77514 on 2003/01/17 by llefebvr@llefebvr_r400_emu_montreal

Z export is now in the RED channel (was previously ALPHA).

Change 77506 on 2003/01/17 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 77498 on 2003/01/17 by rramsey@RRAMSEY_P4_r400_win

Incorporate RTS dump routines into emulator, don't need to run create_rts_dumps.pl
anymore
Fix readback of pa_sc_fifo_size to only return lower 16 bits

Remove call to create_rts_dumps.pl from run_vsim scripts

Fix prints in qdpr_proc/out_compare to have identify correct tracker

Fix a problem with stipple rpt cnt loads in r400sc_rand
Add runtime comment to rand_r400sc.sh

Change 77479 on 2003/01/17 by ashishs@fl_ashishs_r400_win

tests guard band clipping

Change 77432 on 2003/01/17 by kevino@kevino_r400_win_marlboro

Manually set pitch in texture_constant to tb_width. Otherwise, it was left at 0.

Change 77402 on 2003/01/17 by ashishs@fl_ashishs_r400_win

test to determine if the clip guard band works properly and that trivial reject works

Change 77380 on 2003/01/17 by ashishs@fl_ashishs_r400_win

uncommented r400cl_gband_tcl_01

Change 77373 on 2003/01/17 by ashishs@fl_ashishs_r400_win

removed class definitions of Vector3, Vector4 and Matrix and changed XDIM YDIM to
256

Change 77367 on 2003/01/17 by llefebvr@llefebvr_r400_emu_montreal

Dot product random tests

Change 77274 on 2003/01/16 by ashishs@fl_ashishs_r400_win

updated

Change 77240 on 2003/01/16 by ashishs@fl_ashishs_r400_win

Primitive types tri list,strip, fan, Wflags, quad list, quad strip and polygon respectively
permuting edgeflag combinations in each of the test as well as polymode POINT fill enabled
with the ucp combinations enabling and disabling

Change 77220 on 2003/01/16 by ygiang@ygiang_r400_pv_marlboro

fixed: some mova test cases for SQ

Change 77190 on 2003/01/16 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 77187 on 2003/01/16 by mkelly@fl_mkelly_r400_win_laptop

Still need to add proper edge gradients for triangle

Change 77151 on 2003/01/16 by mdoggett@mdoggett_r400_linux_local

Updated to new framebuffer setup.

Change 77131 on 2003/01/16 by omesh@omesh_r400_linux_marlboro_only_devel

Added a basic ZPASS_COUNT test functionality. Currently, the ZPASS_COUNT

does not get written to memory at the address being pointed to, although
the counter was found to be working correctly by Larry S.

Change 77102 on 2003/01/16 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 77100 on 2003/01/16 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 77086 on 2003/01/16 by mkelly@fl_mkelly_r400_win_laptop

12 non real time packets of one triangle, each with 16 real time rectangle streams...

Change 77015 on 2003/01/15 by georgev@devel_georgev_r400_lin2_marlboro

Fixed equal and not equal tests.

Change 77003 on 2003/01/15 by ashishs@fl_ashishs_r400_win

changed 8 textures..

Change 76961 on 2003/01/15 by ashishs@fl_ashishs_r400_win

simple triangle gband clipping test

Change 76926 on 2003/01/15 by ashishs@fl_ashishs_r400_win

gband point culling test converted from R300

Change 76890 on 2003/01/15 by mkelly@fl_mkelly_r400_win_laptop

Fixed viewport settings, changed to PrimLib Color_Surface and other Surface classes

Change 76888 on 2003/01/15 by ashishs@fl_ashishs_r400_win

initial checkin (same as inf_nan_01 except triangle_list insted of point_list)

Change 76878 on 2003/01/15 by vgoel@fl_vgoel2

updated viewport offset to align with pixel position

Change 76844 on 2003/01/15 by mkelly@fl_mkelly_r400_win_laptop

RB bug

Change 76795 on 2003/01/15 by georgev@devel_georgev_r400_lin2_marlboro

Added some new alpha kill tests (not done yet).

Change 76786 on 2003/01/15 by ashishs@fl_ashishs_r400_win

single packet containing around 175 verts and w0 set to true

Change 76716 on 2003/01/14 by ashishs@fl_ashishs_r400_win

To test DX/OGL clip space

Change 76636 on 2003/01/14 by omesh@omesh_r400_linux_marlboro_only_devel

Added tests for using the Export Z path. I ran the "standard" testcase and it didn't seem to produce
the right result, so I will file a bug on the emulator.

Change 76626 on 2003/01/14 by viviana@viviana_crayola_linux_orl

Changed the test to do a GFX_COPY_STATE write between context switching.

Change 76611 on 2003/01/14 by ashishs@fl_ashishs_r400_win

tests barycentric proportions generated by the clipper

Change 76570 on 2003/01/14 by ashishs@fl_ashishs_r400_win

to test barycentrics generated by the clipper

Change 76566 on 2003/01/14 by ashishs@fl_ashishs_r400_win

to test the braycentrics generated by the clipper

Change 76557 on 2003/01/14 by mkelly@fl_mkelly_r400_win_laptop

OGL Rasterization validation...

Change 76536 on 2003/01/14 by mkelly@fl_mkelly_r400_win_laptop

Validate OGL rasterization rules...

Change 76488 on 2003/01/14 by ygiang@ygiang_r400_pv_marlboro

addedL test for debug

Change 76480 on 2003/01/14 by ashishs@fl_ashishs_r400_win

updated comment

Change 76479 on 2003/01/14 by ashishs@fl_ashishs_r400_win

enabling/disabling the vte controls for scale/offset as well as changing offsets/scale with randomised sizes for triangle list packets.

Change 76463 on 2003/01/14 by mkelly@fl_mkelly_r400_win_laptop

OGL rasterization rule check, simple test...

Change 76452 on 2003/01/14 by mkelly@fl_mkelly_r400_win_laptop

Permutations of r400sc_pinwheel_01 where the size,
rotation, and location of the triangles are varied.

Change 76428 on 2003/01/14 by viviana@viviana_crayola_linux_orl

Added the r400su_simple_register_indirect.cpp to test all reads/writes for the context
as well as non-context registers.

Change 76427 on 2003/01/14 by viviana@viviana_crayola_linux_orl

Tests all the context registers for the pa, as well as non-context registers.

Change 76424 on 2003/01/14 by mkelly@fl_mkelly_r400_win_laptop

DX9 rasterization rules check, simple but necessary to confirm...

Change 76422 on 2003/01/14 by mkelly@fl_mkelly_r400_win_laptop

Simple DX rasterization check...

Change 76417 on 2003/01/14 by mkelly@fl_mkelly_r400_win_laptop

Simple test, validate DX rasterization rules...

Change 76412 on 2003/01/14 by mkelly@fl_mkelly_r400_win_laptop

New...

Change 76286 on 2003/01/13 by vgoel@fl_vgoel2

added test with quadstrip with tessellation on.

Change 76268 on 2003/01/13 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 76262 on 2003/01/13 by mkelly@fl_mkelly_r400_win_laptop

Update comments...

Change 76256 on 2003/01/13 by mkelly@fl_mkelly_r400_win_laptop

Final

Change 76246 on 2003/01/13 by ashishs@fl_ashishs_r400_win

Set MSB of multipliers: set all three of XSCALE, x and w to 0.9999998
(don't enable W0_FMT). Repeat for YSCALE, y, and w and ZSCALE, z, and w

Change 76225 on 2003/01/13 by vgoel@fl_vgoel2

added test for 15.0f tessellation

Change 76185 on 2003/01/13 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 76144 on 2003/01/13 by viviana@viviana_crayola_linux_orl

Added rbbm read back test for every context of the context registers.

Change 76133 on 2003/01/13 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 76120 on 2003/01/13 by ashishs@fl_ashishs_r400_win

updated

Change 76118 on 2003/01/13 by ashishs@fl_ashishs_r400_win

updated

Change 76012 on 2003/01/11 by markf@markf_r400_linux_marlboro

Fixed fast depth clear

Change 75949 on 2003/01/10 by csampayo@fl_csampayo_r400

Adding new VGT test, updated test_list

Change 75942 on 2003/01/10 by ashishs@fl_ashishs_r400_win

another vte test

Change 75937 on 2003/01/10 by ashishs@fl_ashishs_r400_win

vte test

Change 75929 on 2003/01/10 by ashishs@fl_ashishs_r400_win

MSB of veu multiplier output set (use X=1.9999, Xscale = 1.9999, and 1/W = 1.9999)

Change 75918 on 2003/01/10 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 75891 on 2003/01/10 by kryan@kryan_r400_win_marlboro

This changelist in PrimLib which allows the FrameBuffer start

to be set to a non-zero value. There are a few caveats with

this which are explained below. Currently I have left

the FrameBuffer start to be zero.

The following code excerpt and comments is from the file

test_lib/src/testchip/chip/init_mcmh.cpp which explains some of the

limitations encountered so far in my testing:

```
const static uint32 FB_START_ADDRESS = 0x00000000;

const static uint32 FB_SIZE          = 0x07FF0000;  // 128M

//-------------------------------------------------------------

// The combination below works for non-zero starting FB,

// but a values causing the top of the FB to be above 64MB

// does not currently seem to work because of problems possibly

// related to the CP not being able to access memory above 64MB

// according to an email from Harry Wise at one time.

//

// It also seems that if the FB_START_ADDRESS is non-zero,
        // then the size should be aligned to 16MB though this is
        // not proven yet, but FB_SIZE of 0x01ff0000 did not work
        // with FB_START_ADDRESS 0x02000000 even though it did
        // satisfy the condition of not creating the top of the FB
        // above 64MB.  This still needs to be investigate, but
        // for now will allow a non-zero FB if it is kept small
        // with the following known working values:
        //      FB_START_ADDRESS = 0x02000000  // 32MB
        //      FB_SIZE          = 0x02000000  // 32MB
        //-------------------------------------------------------------

        // const static uint32 FB_START_ADDRESS = 0x02000000;  // 32M
        // const static uint32 FB_SIZE          = 0x02000000;  // 32M

        MH.FB_START.write( FB_START_ADDRESS );
        MH.HDP_FB_START.write( FB_START_ADDRESS );

        MH.ROM_START.write(0xF0000000); //Put ROM_START way up above everything else for now

        //    MH.FB_SIZE.write( memTop <<16); // FB Size determined by arch.
        MH.FB_SIZE.write( FB_SIZE );   //Hardcoded for now....don't know what this should be
```

In order for all of the Emulator regression tests to pass, I had to modify a few of the tests that fall into the following categories which will also hold true for other tests outside of the Emulator suite that start failing:

1) Tests using Depth Surfaces

Need to be modified to add the start of the FrameBuffer to the Z_BASE offset to calculate the correct device address for the Depth Surface set in the RB_DEPTH_INFO register.

See the tests below for code examples in the changelist above.

gfx/sc/r400sc_poly_offset_05.cpp
gfx/sc/r400sc_poly_offset_fc_02.cpp
gfx/sc/r400sc_zbuffer_list_rectangle_fc_02.cpp

2) Tests using multiple color buffers (but really all tests)

All tests should use the following function RENDER_STATE::Set_Destination_Base(uint32 base, uint32 index) to properly set the base of the Color Surface (taking into account the start of the memory such as FrameBuffer.)

See the tests below for code examples in the changelist above.

gfx/rb/r400rb_mask_color_bits.cpp
gfx/rb/r400rb_mask_color_channels.cpp
gfx/rb/r400_multiwrites.cpp

3) Any test that does not use the Set_Destination_Base() function described above

Any test that directly sets the RB_COLORx_BASE registers needs to

add the start of the FrameBuffer memory area to the offset in the test so that it is calculating the correct device address for this register field.

The code would be as follows:

```
uint32 device_address = frame_buffer.Get_Start() + DISP_BASE;
render_state.Set_RB_COLOR0_BASE_color0_base( device_address );
```

4) Note that this does not take care of the PM4LIB allocating its buffers in the same memory space (ie. same offset in the FrameBuffer as the PrimLib automatic memory management for things like Vertex Buffers, Textures, Shader Programs, etc.) This code still needs to be added later.

The only case that should see this problem is a PrimLib test that is using almost all of the FrameBuffer for its surfaces (or which is manually allocating memory from the top of the FrameBuffer) which may cause the PM4LIB buffers (which start just below the top of the FrameBuffer) to be overwritten. Both of these cases should be rare if non-existent for now. After the code mentioned in 4) is added, then there should not be a problem when the PM4LIB memory management is merged with the PrimLib automatic memory management.

5) gfx/sys/cp tests that use plgx::fill*(), plgx::dump() routines

Since almost all of these tests use the plgx::fill*() routines which expect an absolute device address, they will all break when the

FB start address is changed to something non-zero. For example if the FB start is moved to 0x02000000, then all the accesses to fill_solid(0, ...) will fall outside of all the known apertures currently.

One way to fix this is to rewrite the plgx::fill*() routines to specify the aperture and an offset or to rewrite the tests to retrieve the start of the FrameBuffer and add it to the values used in the test so they still fall inside the FrameBuffer when it is relocated to a non-zero start value.

Another method would be to rewrite these routines to direct all accesses that fall outside any of the apertures to be treated as an offset into the FrameBuffer. This would probably fix these tests, but may cause other problems.

But at least this should allow Toronto to start some testing with a non-zero FrameBuffer start as long as the guidelines above are followed for the short-term.

PM4LIB

- Began preliminary ground work for modify PM4LIB code to use the PrimLib automatic memory management routines. Still not implemented, but began adding code which will be added to later.

MEMORY_FRAMEBUFFER

- Created global object to represent FrameBuffer memory space to store all memory managed in the FrameBuffer by all PrimLib clients.

FRAMEBUFFER_MEMORY

- Modified to just return a pointer to the global FB object so that current code will always be referencing same FB object.

AGP_MEMORY

- Modified Set_Size() function to take a uint32 size and then validate it rather than creating an enum for every possible size.

fill_dump.cpp

- Replaced direct register access code in fill_data(), fill_solid(), dump_image() with appropriate function calls for accessing start, size of AGP memory.

init_mcmh.cpp

- Use variables for FB start and size values.
- Add comments explaining caveats mentioned at top of message in code excerpt.

COLOR_SURFACE

PIXEL_SURFACE

- Modify Dump() routines to use relative addresses from MEMORY_AREA rather than absolute device addresses for surfaces.

MANAGED_MEM_AREA

MANAGED_MEM_BLOCK

- Add Reset_Free_Space_Pointer() function to initialize free space

if start/size of MEMORY_AREA is changed.

MEMORY_AREA

- Added function Is_Memory_Aperture_Inside_Device_Address_Space() to check if MEMORY_AREA is relocated using Set_Size() functions that it is still inside the device address space.

primlib_template_simple_triangle.cpp

- Begin groundwork for accesing AGP memory. Currently still uses FrameBuffer.

Thanks,

Kevin

Change 75875 on 2003/01/10 by viviana@viviana_crayola_linux_orl

Added test r400vgt_simple_register_indirect.cpp.

Change 75861 on 2003/01/10 by georgev@devel_georgev_r400_lin2_marlboro

Added stacked call returns.

Change 75839 on 2003/01/10 by viviana@viviana_crayola_linux_orl

Added a test to write and read all context registers and changed the vgttbtrk_rbbmrd to handle
all contexts.

Change 75820 on 2003/01/10 by ashishs@fl_ashishs_r400_win

1 Point , cycling Infinity/NANS XYZ scale and Infinity/NANS XYZ offset through all bits including sign changes

Change 75784 on 2003/01/10 by kevino@kevino_r400_win_marlboro

All of the _on_2x2 cases were actually using 3x3 prims, so fixed.

Change 75772 on 2003/01/10 by mdoggett@mdoggett_r400_linux_local

Updated test structure to use latest version of primlib_tex_tri.

Change 75764 on 2003/01/10 by ashishs@fl_ashishs_r400_win

initial checkin

Change 75726 on 2003/01/10 by mkelly@fl_mkelly_r400_win_laptop

Change test search bug string from "bug" to "_bug"

Change 75665 on 2003/01/09 by csampayo@fl_csampayo_r400

Adding another RTS test

Change 75662 on 2003/01/09 by georgev@devel_georgev_r400_lin2_marlboro

Added new tests.

Change 75658 on 2003/01/09 by markf@markf_r400_linux_marlboro

Added some more cases

Change 75642 on 2003/01/09 by csampayo@fl_csampayo_r400

Added 1 VGT performance and 1 VGT debug case, updated test_list and tracker accordingly

Change 75618 on 2003/01/09 by mdoggett@mdoggett_r400_linux_local

Changed temporary setting of format to 16_16_16_16 to correct DXT1.

Change 75606 on 2003/01/09 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 75533 on 2003/01/09 by ashishs@fl_ashishs_r400_win

added a generalised Matrix class to the test used to calculate the rotations.

Change 75528 on 2003/01/09 by mkelly@fl_mkelly_r400_win_laptop

Finalize comments...

Change 75492 on 2003/01/09 by mkelly@fl_mkelly_r400_win_laptop

Update RB setup using r400rb_tb.cpp as an example...

Change 75450 on 2003/01/09 by mkelly@fl_mkelly_r400_win_laptop

Update comments in test...

Change 75439 on 2003/01/09 by smoss@smoss_crayola_linux_orl_regress

<Orlando Hardware Regression Results >

Change 75384 on 2003/01/08 by ashishs@fl_ashishs_r400_win

initial checkin

Change 75383 on 2003/01/08 by ashishs@fl_ashishs_r400_win

initial checkin

Change 75340 on 2003/01/08 by vromaker@vromaker_r400_linux_marlboro

fix for NOP in CFS; reduced triangle size in sq_tests

Change 75334 on 2003/01/08 by omesh@omesh_r400_linux_marlboro_only_devel

Fixed compile time bugs introduced in merge by diagnostics.

Change 75321 on 2003/01/08 by markf@markf_r400_linux_marlboro

Simple texture fill rate test

Change 75305 on 2003/01/08 by llefebvr@llefebvre_laptop_r400_emu

Now using the new mulAdd function.

Change 75303 on 2003/01/08 by mkelly@fl_mkelly_r400_win_laptop

Stress vtx and pix pipe disable combinations with stippled LINE_LIST

Change 75291 on 2003/01/08 by vgoel@fl_vgoel2

changed maximum allowable tessellation to 15.0f.

Change 75282 on 2003/01/08 by pauld@pauld_r400_win_marlboro

updates for diagnostic environment

Change 75233 on 2003/01/08 by smoss@smoss_crayola_linux_orl_regress

moved to crayola2

Change 75190 on 2003/01/08 by ashishs@fl_ashishs_r400_win

point with 8 textures. added just for own reference

Change 75173 on 2003/01/08 by csampayo@fl_csampayo_r400

Added new VGT pass-thru block test, updated test_list and test tracker accordingly.

Change 75168 on 2003/01/08 by ashishs@fl_ashishs_r400_win

vertex and pixel shaders for r400cl_point_size_ucp_combo_01 test(to switch between point size in vertex export and SU state registers)

Change 75162 on 2003/01/08 by mkelly@fl_mkelly_r400_win_laptop

This test has changed to r400sc_vtx_pipe_disable_combos_03

Change 75149 on 2003/01/08 by lseiler@lseiler_r400_win_marlboro

Blender now uses hardware precision in most places -- resulted in one pixel difference in r400rb_color_source

Change 75086 on 2003/01/07 by vgoel@fl_vgoel2

change for modified vertex export

Change 75049 on 2003/01/07 by ygiang@ygiang_r400_win_marlboro_p4

modifyied for debugging

Change 75040 on 2003/01/07 by ashishs@fl_ashishs_r400_win

increased the xdim and ydim for better visibility

Change 75036 on 2003/01/07 by ashishs@fl_ashishs_r400_win

verify the 64 combinations of ucp control bits with quadstrip primitive

Change 75024 on 2003/01/07 by mkelly@fl_mkelly_r400_win_laptop

Update..

Change 75017 on 2003/01/07 by ashishs@fl_ashishs_r400_win

verify the 64 combinations of ucp control bits with polygon primitive

Change 75003 on 2003/01/07 by mkelly@fl_mkelly_r400_win_laptop

Temporarily remove msaa

Change 74990 on 2003/01/07 by mkelly@fl_mkelly_r400_win_laptop

Temporarily remove msaa

Change 74925 on 2003/01/07 by csampayo@fl_csampayo2_r400

Updated for latest RB surface definitions

Change 74913 on 2003/01/07 by hartogs@fl_hartogs2

Added VgtGrpOut.dmp

Change 74910 on 2003/01/07 by mkelly@fl_mkelly_r400_win_laptop

Temporarily removed MSAA

Change 74897 on 2003/01/07 by hartogs@fl_hartogs2

Added new tracker file "VgtGrpOut.dmp"

Change 74895 on 2003/01/07 by kevino@kevino_r400_win_marlboro

Added mipmap test that uses small textures and prims. Added to tp4_tc nightly regressions as well.

Change 74894 on 2003/01/07 by ashishs@fl_ashishs_r400_win

creating 11/12/13/14/15 vertices from 8/9/10/11/12 clip planes respectively

Change 74890 on 2003/01/07 by llefebvr@llefebvre_laptop_r400_emu

Nes template for the memory exports. This is to include the fact that ARRAY_2D is invalif for memory exports AND changes the size of endian to 3 bits.

Change 74883 on 2003/01/07 by kevino@kevino_r400_win_marlboro

MaxMipLevel was=0, so never mipmapped. Fixed this and added equivilant test cases with the mip filter set to BaseMap.

Change 74873 on 2003/01/07 by mkelly@fl_mkelly_r400_win_laptop

Two cases isolated for hw debug assistance...

Change 74869 on 2003/01/07 by kevino@kevino_r400_win_marlboro

modified to use buildlevel for 3D textures

Change 74848 on 2003/01/07 by hartogs@fl_hartogs

Added vgt_grouper output dump for RTL comparison.
Removed stride==0 as a legitimate value for grouper programming. Use fully overlapping vectors instead.

Change 74799 on 2003/01/06 by ygiang@ygiang_r400_linux_marlboro

fixed: time out problem for random test case

Change 74788 on 2003/01/06 by pauld@pauld_r400_win_marlboro

files updated for diagnostic environment

Change 74785 on 2003/01/06 by pauld@pauld_r400_win_marlboro

files updated for diagnostic environment

Change 74773 on 2003/01/06 by ashishs@fl_ashishs_r400_win

different settings for the cube position than cube_01.cpp and enabling all the 6 user clip planes. Also the culling direction is reversed.

Change 74760 on 2003/01/06 by pauld@pauld_r400_win_marlboro

fix merge errors adding diag environment

Change 74739 on 2003/01/06 by ashishs@fl_ashishs_r400_win

updated to have exact same settings as on R200 legacy test

Change 74708 on 2003/01/06 by pauld@pauld_r400_win_marlboro

r400rb files updated dor diagnostic environment

Change 74697 on 2003/01/06 by ashishs@fl_ashishs_r400_win

updated

Change 74667 on 2003/01/06 by vgoel@fl_vgoel2

simplified further for bug tracing

Change 74665 on 2003/01/06 by vgoel@fl_vgoel2

changed for bug tracing

Change 74644 on 2003/01/06 by ashishs@fl_ashishs_r400_win

To test user clip plane in clip space
A cube is rotated over x and y axis to get a perspective view of the cube and then the cube is clipped with a UCP plane which is parallel to the XY plane and goes through point (0,0,0) The UCP then clips the cube as required. The test also tests the SU_SC_MODE_CNTL register
with turning ON the front face culling and turning OFF the back face culling.

Change 74602 on 2003/01/06 by mkelly@fl_mkelly_r400_win_laptop

Stress vtx and pix pipe disable in SC...

Change 74446 on 2003/01/03 by mkelly@fl_mkelly_r400_win_laptop

Random pix and vtx pipe disable combos...

Change 74426 on 2003/01/03 by mkelly@fl_mkelly_r400_win_laptop

VTX and PIX pipe disable combinations

Change 74420 on 2003/01/03 by mkelly@fl_mkelly_r400_win_laptop

Pixel pipe disable combinations...

Change 74415 on 2003/01/03 by omesh@omesh_r400_linux_marlboro_only_devel

Fixed the problem with the test which fixed Bugzilla #951. Even though the pixel center was set to the center of the grid, I was sending down vertex coordinates as if I were rendering with the pixel center at the top-left of the grid.

Change 74414 on 2003/01/03 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 74411 on 2003/01/03 by mkelly@fl_mkelly_r400_win_laptop

Make Set_VGT_OUT_DEALLOC_CNTL_dealloc_dist = Set_VGT_VERTEX_REUSE_BLOCK_CNTL_vtx_reuse_depth

Change 74401 on 2003/01/03 by mkelly@fl_mkelly_r400_win_laptop

Initialize MC disable field to zero.

Change 74383 on 2003/01/03 by ashishs@fl_ashishs_r400_win

Point sprite frustum culling. This test is intended to validate the shader processing of point sprite primitives with frustum culling.

Change 74369 on 2003/01/03 by omesh@omesh_r400_linux_marlboro_only_devel

Fixed a typo. The OnePixel8Fragment case still doesn't hit the right samples.

Change 74366 on 2003/01/03 by mkelly@fl_mkelly_r400_win_laptop

Add initialization of the MC disable field.

Change 74357 on 2003/01/03 by mkelly@fl_mkelly_r400_win_laptop

256 points per packet, all 15 legal combinations of vtx pipe disable

Change 74342 on 2003/01/03 by kevino@kevino_r400_win_marlboro

check_tfc_overrides added. If it is a float format, it overrides any linear filters with point.

Change 74330 on 2003/01/03 by mkelly@fl_mkelly_r400_win_laptop

Fix a bug for the case of sequential tests in test_list which use agp_r for compare.

Change 74323 on 2003/01/03 by ashishs@fl_ashishs_r400_win

This test is intended to validate the vertex reuse functionality with actual clipping. The test processes 56 packets each with either one 100 primitive triangle list or one 150 primitive line list containing 16 vertices with input vertex data: XYZW0, 1 color , no textures. For each packet the test has 300 indices, making up either 100 or 150 primitives, depending on randomly selected primitive type, unique indices are randomly selected between 0-12 and 0-15. For each packet six UCP planes are set up so that most of the original verices get clipped.

Change 74303 on 2003/01/03 by ashishs@fl_ashishs_r400_win

Test processes 64 packets each a set of tri-lists, each with 15 triangles using 10 vertices

Change 74232 on 2003/01/02 by georgev@devel_georgev_r400_lin2_marlboro

Added shader program for existing test.

Change 74223 on 2003/01/02 by kryan@kryan_r400_win_marlboro

Modify test to align pitch and height of texture dump to 32 pixels to avoid error for

non-power of 2 textures.

Change 74184 on 2003/01/02 by csampayo@fl_csampayo_lt_r400

Updated test and test_list for test r400vgt_real_time_events_06 and updated description/status on the test tracker for tests:
r400vgt_real_time_events_04
r400vgt_real_time_events_05
r400vgt_real_time_events_06

Change 74176 on 2003/01/02 by csampayo@fl_csampayo2_r400

Initial check-in

Change 74146 on 2003/01/02 by viviana@viviana_crayola_linux_orl

Test to write and read the rbbm/pa registers.

Change 74131 on 2003/01/02 by omesh@omesh_r400_linux_marlboro_only_devel

Added some PA programming that might have caused changes in vertex X,Y
data entering the SC. It turns out that this programming didn't change
bug symptom Bugzilla#951

Change 74126 on 2003/01/02 by mkelly@fl_mkelly_r400_win_laptop

Duplicate coverage, remove tests...

Change 74122 on 2003/01/02 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 74112 on 2003/01/02 by mkelly@fl_mkelly_r400_win_laptop

Vary levels of tesselation, rasterize such that each case can be seen
in the framebuffer.

Change 74101 on 2003/01/02 by viviana@viviana_crayola_linux_orl

Added r400su_rbbm_reg_read to the list

Change 74100 on 2003/01/02 by viviana@viviana_crayola_linux_orl

Added r400sc_rbbm_reg_read to the test_list.

Change 74096 on 2003/01/02 by hartogs@fl_hartogs

Added Vivian's test "r400vgt_rbbm_reg_read"

Change 74092 on 2003/01/02 by mkelly@fl_mkelly_r400_win_laptop

Increased framebuffer size from 64x64 to 256x256.
Rasterize each case, XY offset by the VTE to display all cases.

Change 74089 on 2003/01/02 by llefebvr@llefebvr_r400_emu

Changed the test to test MUL_PREV2, was checking MUL_PREV

Change 73945 on 2002/12/31 by jhoule@jhoule_r400_ma-jhoule-linux

LoadDefaultTexture no longer sends tiled 1D textures

Change 73939 on 2002/12/31 by jhoule@jhoule_r400_ma-jhoule-linux

Forces linear for 1D textures in LoadDefaultTexture.

Change 73933 on 2002/12/31 by pmitchel@regress_test_r400_linux_marlboro

makefile support for regressing and releasing parameterized tests/testcases
see 'make help' from underneath test_lib for details
'fixes' bug 741

Change 73918 on 2002/12/31 by mkelly@fl_mkelly_r400_win_laptop

HOS continue flag indices with vtx pipe disable combinations...

Change 73915 on 2002/12/31 by mkelly@fl_mkelly_r400_win_laptop

All vtx pipe disable combos...

Change 73906 on 2002/12/31 by mkelly@fl_mkelly_r400_win_laptop

Vertex shader outputs two vectors to PA with vtx pipes 2 and 3 disabled.

Change 73896 on 2002/12/31 by mkelly@fl_mkelly_r400_win_laptop

Update for new XY handling...

Change 73891 on 2002/12/31 by smoss@smoss_crayola_linux_orl_regress

increased timeout

Change 73852 on 2002/12/31 by mkelly@fl_mkelly_r400_win_laptop

Centers and Centroids state switching

Change 73841 on 2002/12/31 by mkelly@fl_mkelly_r400_win_laptop

Change...

Change 73840 on 2002/12/31 by mkelly@fl_mkelly_r400_win_laptop

Invalid test, delete...

Change 73765 on 2002/12/30 by ashishs@fl_ashishs_r400_win

error in description. changed the description to as follows. test setup same as earlier tests.
THE POINT IS RETAINED WHNEVER THE CENTER OF POINT IS INSIDE THE CLIP PLANE. THE POINT IS THROWN AWAY WHENEVER THE CENTER OF POINT LIES OUTSIDE THE CLIP PLANE.

Change 73764 on 2002/12/30 by ashishs@fl_ashishs_r400_win

SETUP WITH PS UCP MODE = 1 (CULL USING RADIUS BASED DISTANCE) and CULL ONLY DISABLED.
RESULT:WHENEVER THE CLIP PLANE IS ON OR WITHIN THE CULL RADIUS OF THE POINT, THE POINT IS CULLED.THE POINT IS THROWN AWAY WHENEVER THE CENTER OF THE POINT LIES OUTSIDE THE CLIP PLANE AND THE DISTANCE BETWEEN THE CENTER OF THE POINT AND CLIP PLANE IS GREATER THAN THE CULL RADIUS OF THE POINT.

Change 73762 on 2002/12/30 by ashishs@fl_ashishs_r400_win

TEST WITH PS MODE AS 0 AND CULL DISABLED:
RESULT:WHENEVER THE CLIP PLANE IS ON OR WITHIN THE HALF WIDTH OF THE POINT, THE POINT IS CULLED.THE POINT IS THROWN AWAY WHENEVER THE CENTER OF THE POINT LIES OUTSIDE THE CLIP PLANE AND THE DISTANCE BETWEEN THE CENTER OF THE POINT AND CLIP PLANE IS GREATER THAN THE HALF WIDTH OF THE POINT.

Change 73754 on 2002/12/30 by ashishs@fl_ashishs_r400_win

setup same as previous tests with UCP point sprite mode to be 3 (ALWAYS CLIP AND EXPAND AS TRIFAN) and cull only disabled.

Change 73753 on 2002/12/30 by mkelly@fl_mkelly_r400_win_laptop

Add write to MC_DISABLE field = 0x0

Change 73750 on 2002/12/30 by ashishs@fl_ashishs_r400_win

the test has same setup as the previous tests in the category. The difference being that the point sprite ucp mode is set to 3 (with cull only enabled). Hence the only difference between the the r400cl_point_ucp_clip_mode2_cull_enable and this test is that the points are always expanded in this test. No matter if the clip plane is within the cull radius of the point the point is always expanded into trifan. Hence the difference in both tests outputs can be seen on Case 11 in both the cases.

Case 11 of the test: The center of point lies INSIDE the clip plane and distance between the center of point and the clip plane = radius of the point + delta

i.e the point is totally inside or doesnt have any thing to do with the clip plane. Hence when the point spirte mode is 2 the point isnt expanded wheras when the point sprite mode is 3 the point is expanded and can be seen with the texture being inverted.

Change 73747 on 2002/12/30 by ashishs@fl_ashishs_r400_win

added more number of cases. generalised this test to be similar to the r400cl_point_ucp_clip_mode2_cull_disable_01.cpp test with cull only enabled.

Change 73745 on 2002/12/30 by mkelly@fl_mkelly_r400_win_laptop

HOS test which generates continue flags and coefficient data in the indice stream from vgt to sq
for pipe disable testing...

Change 73744 on 2002/12/30 by kryan@kryan_r400_win_marlboro

- Missed this file in the check for changelist 73741

- Changed to set height to 0 when dumping 1D textures.

Change 73741 on 2002/12/30 by kryan@kryan_r400_win_marlboro

MEMORY_AREA

- Modify Dump(...) functions to use height parameter to determine

if surface is 1D, or 2D. This dimension is then passed to the

surface constructor, and will not check height constraints for

a 1D surface such as a 1D texture being dumped.

tp_unsigned_01.cpp

tp_unsigned_01_stmap.cpp

- Modify tests to pass height of 0 when dumping 1D textures. This

eliminates the error from PrimLib previously because it was

considering all surfaces being dumped to be 2D and therefore

was checking the height constraint which was 1 for 1D textures.

Change 73721 on 2002/12/30 by ashishs@fl_ashishs_r400_win

The purpose of this test is to test ucp clip for point list with
UCP Point Sprite Mode = 2(CULL USING RADIUS BASED DISTANCE FROM CENTER OF THE POINT)
with ucp cull only disabled.

THE RESULT CAN BE SUMMARIZED AS FOLLOWS:
WHENEVER THE CLIP PLANE IS ON OR WITHIN THE CULL RADIUS OF THE POINT, THE POINT IS EXPANDED. THE POINT IS CLIPPED ON INTERSECTION WITH THE CLIP PLANE(MEANING THE POINT IS THROWN OUT IF THE POINT EDGES

i.e.HALF WIDTH ARE OUTSIDE THE CLIP PLANE EVEN IF THE CLIP PLANE IS WITHIN THE CULL RADIUS OF THE POINT AND CLIPPED TO PROPER DIMENSIONS ON INTERSECTION OF THE POINT BOUNDARIES WITH THE CLIP PLANE.)

MORE IN TEST DESCRIPTION

Change 73688 on 2002/12/30 by markf@markf_r400_win_marlboro

Added test for scalar add

Change 73683 on 2002/12/30 by mkelly@fl_mkelly_r400_win_laptop

Two 96bit transfers per vert...

Change 73527 on 2002/12/27 by mkelly@fl_mkelly_r400_win_laptop

Basic MSAA 8 test, need to get it to work on GC

Change 73515 on 2002/12/27 by markf@markf_r400_win_marlboro

Added random testing of mul add precison

Change 73502 on 2002/12/27 by llefebvr@llefebvre_laptop_r400_emu

Changing the XY import to the SP as per Tom Frinsinger's proposal.

Change 73498 on 2002/12/27 by omesh@omesh_r400_linux_marlboro_only_devel

Added the Degamma test with 22 testcases. Ran them, but haven't verified
emulator image output yet, as DumpView wasn't working.

Change 73496 on 2002/12/27 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 73424 on 2002/12/26 by mkelly@fl_mkelly_r400_win_laptop

Update tracker...

Change 73417 on 2002/12/26 by mkelly@fl_mkelly_r400_win_laptop

Enable RTS in test...

Change 73254 on 2002/12/23 by abeaudin@abeaudin_r400_win_marlboro

fixed hardware precison problem

Change 73241 on 2002/12/23 by smoss@smoss_crayola_linux_orl_regress

increased timeout

Change 73220 on 2002/12/23 by mkelly@fl_mkelly_r400_win_laptop

Fix some test bugs, first cut at rts with SC packer optimize...

Change 73127 on 2002/12/23 by mkelly@fl_mkelly_r400_win_laptop

SC packer optimize with pipe disable variations...

Change 73123 on 2002/12/23 by mkelly@fl_mkelly_r400_win_laptop

Test alternate bits compare to r400sc_viz_query_01

Change 72996 on 2002/12/21 by jhoule@jhoule_r400_win_marlboro

Support for LOD_BIAS while keeping the old ones (_H/_V)

Change 72946 on 2002/12/20 by kryan@kryan_r400_win_marlboro

COLOR_SURFACE

- Remove register reads from .rd_r file for all PrimLib Dump()

function calls.

MEMORY_AREA

- Modify Dump(..., plgx::pixelType, ...) function to go through one

common function for all pixel types. This way the header

will be the same for all dumps.

chip/gfx/rb/

- Update golden images for .rd_r files for two tests since

register reads are no longer sent to this file.

DEPTH_SURFACE

- Modify field names in header file for clarity.

Change 72873 on 2002/12/20 by omesh@ma_omesh

Made fix to revision 11 to fix compile error on Windows.

Change 72868 on 2002/12/20 by mkelly@fl_mkelly_r400_win_laptop

Complete, stress SC quad packer for coverage, including back pressure
and specific pattern insertion of 1qd, 4qd, 2qd, 4qd, 3qd, 4qd

Change 72858 on 2002/12/20 by omesh@omesh_r400_linux_marlboro_only_devel

Added the PA_SC_AA_CONFIG.MAX_SAMPLE_DIST programming but bug symptoms
still remain.

Change 72828 on 2002/12/20 by georgev@devel_georgev_r400_lin2_marlboro

Added back test and removed errant instructions.

Change 72775 on 2002/12/20 by georgev@devel_georgev_r400_lin2_marlboro

Added simple jumps (really this time).

Change 72710 on 2002/12/20 by smoss@smoss_crayola_win

SU tests

Change 72707 on 2002/12/20 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 72654 on 2002/12/19 by csampayo@fl_csampayo2_r400

Adding 2 new realtime streams tests, updated test_list accordingly

Change 72649 on 2002/12/19 by markf@markf_r400_win_marlboro

Added some cases that only do clears w/ no drawing

Change 72609 on 2002/12/19 by jhoule@jhoule_r400_win_marlboro

Added support for volume maps and perspective correction.

Change 72564 on 2002/12/19 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 72545 on 2002/12/19 by ashishs@fl_ashishs_r400_win

updated to have correct mode

Change 72537 on 2002/12/19 by omesh@omesh_r400_linux_marlboro_only_devel

Changed test to use rectangle primitives instead of quad primitives, to
offer a cleaner test stimulus to the RBC interface for Bill.

Change 72535 on 2002/12/19 by vgoel@fl_vgoel2

simplified the test

Change 72527 on 2002/12/19 by smoss@smoss_crayola_win

SU tests

Change 72474 on 2002/12/19 by omesh@omesh_r400_linux_marlboro_only_devel

Changed (fixed) factor for numerical range for the SINTEGER and UINTEGER cases to use the entire range.

Change 72451 on 2002/12/19 by csampayo@fl_csampayo_lt_r400

Correct shader file names

Change 72426 on 2002/12/19 by smoss@smoss_crayola_linux_orl_regress

test was programming the fifo to a value larger than designed

Change 72417 on 2002/12/19 by georgev@devel_georgev_r400_lin2_marlboro

Fixed tests.

Change 72387 on 2002/12/19 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 72386 on 2002/12/19 by mkelly@fl_mkelly_r400_win_laptop

basic pipe disable configurations matching vtx and pix

Change 72367 on 2002/12/19 by mkelly@fl_mkelly_r400_win_laptop

Pipe disable vtx 0 pix 0

Change 72365 on 2002/12/19 by mkelly@fl_mkelly_r400_win_laptop

simple triangle as a baseline for some other tests...

Change 72317 on 2002/12/18 by csampayo@fl_csampayo2_r400

Update to level at csampayo2

Change 72295 on 2002/12/18 by csampayo@fl_csampayo_lt_r400

Update frame buffer sizes

Change 72250 on 2002/12/18 by omesh@omesh_r400_linux_marlboro_only_devel

Fixed test.... Reduced the "all" and "fog_random" testcases exactly by a factor of 16 in terms of
number of pixels rendered.

Change 72212 on 2002/12/18 by pauld@pauld_r400_win_marlboro

test modified for Diag Environment, Watcom Compiler

Change 72182 on 2002/12/18 by ashishs@fl_ashishs_r400_win

checkpoint for left ucp plane

Change 72161 on 2002/12/18 by grayc@chip_regress_orl

temp compile script

Change 72149 on 2002/12/18 by smoss@smoss_crayola_win

SU tests

Change 72146 on 2002/12/18 by pauld@pauld_r400_win_marlboro

tests modified for Diag Environment, Watcom Compiler

Change 72144 on 2002/12/18 by mkelly@fl_mkelly_r400_win_laptop

Test SC FIFO sizing...

Change 72133 on 2002/12/18 by csampayo@fl_csampayo3_r400

Uncomment tests that now run ok

Change 72132 on 2002/12/18 by markf@markf_r400_win_marlboro

Pass through shaders for a few tests

Change 72130 on 2002/12/18 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 72128 on 2002/12/18 by mkelly@fl_mkelly_r400_win_laptop

Initial test, disable pix and vtx pipe 0

Change 72094 on 2002/12/18 by csampayo@fl_csampayo3_r400

Delete currently unavailable test r400vgt_real_time_events_04

Change 72082 on 2002/12/18 by csampayo@fl_csampayo_r400

Updated test status and test_list for the following tests:
r400cl_clip_edgeflags_frustum_corners_01
r400cl_clip_edgeflags_frustum_corners_02

Change 72075 on 2002/12/18 by mkelly@fl_mkelly_r400_win_laptop

Changed to one read at end of viz query tests...

Change 72068 on 2002/12/18 by mkelly@fl_mkelly_r400_win_laptop

Test PA_CL_ENHANCE, vertex reorder and clip seq random combinations...

Change 71995 on 2002/12/17 by ashishs@fl_ashishs_r400_win

initial checkin for the point sprite mode = 2 (CULL USING RADIUS BASED DISTANCE FROM CENTER POINT EXPAND CLIP ON INTERSECTION) with cull only enable = true

Change 71988 on 2002/12/17 by ashishs@fl_ashishs_r400_win

The purpose of this test is to test ucp clip for point list with UCP Point Sprite Mode = 0(CULL USING DISTANCE FROM CENTER OF THE POINT)

The test generates 12 textured points in an orderly fashion around the 4 enabled ucp planes i.e (top,left,right,bottom gband edge) it generates 3 points on each ucp plane. Out of the 3 points, 1 point lies eactly on the ucp plane, the other 2 points slightly outside and slightly inside(+ or - 0.000001f with respect to the ucp plane)
When Clipping is enabled for ucp planes, the test retains all the points lying ON or INDISE the ucp planes wheras discards all the points lying OUTSIDE the ucp planes.

Change 71966 on 2002/12/17 by csampayo@fl_csampayo_r400

Adding new clipper tests checking polymode corner cases

Change 71914 on 2002/12/17 by lseiler@lseiler_r400_win_marlboro2

Added new endian modes to dumpview and rb color format tests

Change 71895 on 2002/12/17 by ashishs@fl_ashishs_r400_win

The purpose of this test is to test ucp clip for point list with UCP Point Sprite Mode = 1(CULL USING RADIUS BASED DISTANCE FROM CENTER OF THE POINT)

The test generates 28 textured points in an orderly fashion around the ucp's which areset to clip like gbands (using 4 ucp's) . i.e (top,left,right,bottom ucp edge)
The test generates 7 points on each ucp plane. Out of the 7 points, 1 point lies eactly on the ucp plane, the other 6 vary based on the cases specified in the test.

The cases are given below (delta = 0.000001f):
Case 1: The center of point lies exactly on the ucp plane.
Case 2: The center of point lies INSIDE the clip plane and distance between the center of point and the clip plane = radius of the point - delta
Case 3: The center of point lies INSIDE the clip plane and distance between the center of point and the clip plane = radius of the point
Case 4: The center of point lies INSIDE the clip plane and distance between the center of point and the clip plane = radius of the point + delta
Case 5: The center of point lies OUTSIDE the clip plane and distance between the center of point and the clip plane = radius of the point - delta
Case 6: The center of point lies OUTSIDE the clip plane and distance between the center of point and the clip plane = radius of the point
Case 7: The center of point lies OUTSIDE the clip plane and distance between the center of point and the clip plane = radius of the point + delta

When Clipping is enabled for ucp's, the test retains all the cases except the case 7

RETAINS THE POINTS WHEN THE DISTANCE BETWEEN THE CENTER OF THE POINT AND THE CLIP PLANE
IS LESS THAN OR EQUAL TO THE RADIUS
DISCARDS THE POINTS WHEN THE DISTANCE BETWEEN THE CENTER OF THE POINT AND CLIP PLANE IS GREATER THAN THE RADIUS AND THE CENTER OF THE POINT IS OUTSIDE THE CLIP PLANE

Change 71857 on 2002/12/17 by omesh@omesh_r400_linux_marlboro_only_devel

Using consolidated register write instead of setting each register field
seperately.

Change 71831 on 2002/12/17 by lseiler@lseiler_r400_win_marlboro2

Updated golden files

Change 71825 on 2002/12/17 by lseiler@lseiler_r400_win_marlboro2

Support two new endian modes 8IN64 and 8IN128, required new golden images for two tests that wrote out register values

Change 71819 on 2002/12/17 by smoss@smoss_crayola_win

SU tests

Change 71767 on 2002/12/17 by omesh@omesh_r400_linux_marlboro_only_devel

Changed test to use quad lists instead of triangle lists to eliminate all rasterization
concerns associated with per pixel multisample rendering. This seems to produce all quad sample masks
correctly, as verified by Bill, although the tile probes now don't seem to work as expected.

Change 71755 on 2002/12/17 by ashishs@fl_ashishs_r400_win

updated to less number of cases using 3 cases for each plane (exactly on the plane, just outside the plane and just inside the plane)

Change 71740 on 2002/12/17 by markf@markf_r400_linux_marlboro

Added standard and standard_z test cases

Change 71734 on 2002/12/17 by omesh@omesh_r400_linux_marlboro_only_devel

Changed test to use quads lists instead of triangle lists to eliminate all rasterization
concerns associated with per pixel multisample rendering. This seems to produce all quad sample masks
correctly, as verified by Bill, although the tile probes now don't seem to work as expected.

Change 71723 on 2002/12/17 by mkelly@fl_mkelly_r400_win_laptop

Robustly check sc_one_quad_per_clock and quad_order_enable...

Change 71716 on 2002/12/17 by ashishs@fl_ashishs_r400_win

updated to less number of cases and decreased the delta for cases that lie inside and outside the gbands

Change 71676 on 2002/12/17 by mkelly@fl_mkelly_r400_win_laptop

Robustly check pkr row disable...

Change 71656 on 2002/12/17 by smoss@smoss_crayola_linux_orl_regress

renamed shader files for unix case

Change 71623 on 2002/12/16 by omesh@omesh_r400_linux_marlboro_only_devel

Added more golden results to list for regress_e. Also added standard testcases to tests that were missing them.

Change 71593 on 2002/12/16 by ygiang@ygiang_r400_linux_marlboro

added: random testing for sp

Change 71588 on 2002/12/16 by markf@markf_r400_lt_marlboro

New simple test for tile buffer debug

Change 71584 on 2002/12/16 by ygiang@ygiang_r400_linux_marlboro

fixed: emu time out

Change 71579 on 2002/12/16 by georgev@devel_georgev_r400_lin2_marlboro

Added new tests and fixed old ones.

Change 71573 on 2002/12/16 by vgoel@fl_vgoel2

added stress test for hos which will use all primtitive and tessellation type. It is not complete yet.

Change 71503 on 2002/12/16 by ashishs@fl_ashishs_r400_win

The purpose of this test is to test frustum clip planes for point list.
The test generates 28 textured points in an orderly fashion around the frustum planes. i.e (top,left,right,bottom frustum edge) it generates 7 points on each plane. Out of the 7 points, 1 point lies eactly on the frustum plane whereas the remainign 6 point positions vary, with 3 points on the inside and 3 points on the outside of the frustum.

This test generates additionally 6 points for the near and the far planes and the clip space is defined to be OGL clip space. Out of the 6 points, 3 points are close to the near plane wheras 3 points are close to the far plane. Similarly out of the 3 points 1 point lies outside,1 point lies on the frustum plane and the 3rd point lies inside the frustum plane.

When Clipping is enabled for frustum, the test retains all the points lying ON or INDISE the frustum wheras discards all the points lying OUTSIDE the frustum viz performs culling

Change 71502 on 2002/12/16 by omesh@omesh_r400_linux_marlboro_only_devel

Added 2 valid testcases using a different way of hitting specific samples in screen space (Not using the PA sample mask).
Even if I render subpixel triangles, the SX->RB quad mask waveforms don't show the right detail masks of the right samples being rendered to.
The only 2 valid testcases in this file are "OnePixel8Fragments" and "OnePixel4thFragment".

Change 71501 on 2002/12/16 by omesh@omesh_r400_linux_marlboro_only_devel

Added missing texture constant programming needed for resolve a few days
ago, but forgot to check this file in. The resolve seems to work, but

the multisampling does not seem to hit the right samples with the right
colors. Will add a Bugzilla by the end of the day after further
investigation.

Change 71468 on 2002/12/16 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 71461 on 2002/12/16 by abeaudin@abeaudin_r400_win_marlboro

fixed blender GL_ONE precision problem

Change 71454 on 2002/12/16 by ashishs@fl_ashishs_r400_win

The purpose of this test is to test gband clip for point list. The test generates 28 textured points in an orderly fashion around the gbands. i.e (top,left,right,bottom gband edge) it generates 7 points on each gband. Out of the 7 points, 1 point lies eactly on the gband whereas the remainign 6 point positions vary with 3 points on the inside and 3 points on the outside of the gband.
When Clipping is enabled for gbands, the test retains all the points
lying ON or INDISE the gbands wheras discards all the points lying OUTSIDE the gbands viz performs culling

Change 71348 on 2002/12/16 by mkelly@fl_mkelly_r400_win_laptop

RealDOT = 1 by default

Change 71174 on 2002/12/13 by csampayo@fl_csampayo2_r400

Added 3 new tests

Change 71173 on 2002/12/13 by csampayo@fl_csampayo2_r400

Adding new SU polymode culling tests

Change 71144 on 2002/12/13 by markf@markf_r400_win_marlboro

yadda, yadda

Change 71126 on 2002/12/13 by georgev@devel_georgev_r400_lin2_marlboro

Added new tests.

Change 71071 on 2002/12/13 by ctaylor@fl_ctaylor_r400_dtwin_marlboro

Add function to RC to allow test-level interface to enable random hi-z kills for random testing.
Added fuction call to r400sc_rand.cpp

Added SERIALIZE to VFD utility to allow auto-gen tests to run both faster and more realistically.

Change 71056 on 2002/12/13 by ctaylor@fl_ctaylor_r400_dtwin_marlboro

re-enable perf counters

Change 70967 on 2002/12/13 by ashishs@fl_ashishs_r400_win

the problem with the earlier checkin solved using
render_engine.Wait_Gfx_Idle_No_Flush(); . Clipping also enabled.

The test currently has a problem in which the some clipped points dont get the correct texture. Need to check if its an aliaising effect.

Change 70958 on 2002/12/13 by ashishs@fl_ashishs_r400_win

The test renders 128 points. 64 points with point size from vertex export and 64 points with point size from state registers of SU alternating each time. Also it varies the point size for the 64 points with point size from vertex export alternating between 2 different point sizes. The points are clipped using 6 UCP clip planes viz 64 combinations thereby generating 64 cases for each of the point size type viz vertex export 64 cases and SU state registers 64 combinations.
The test also verifies the point sprite texturing mechanism built into the SU/SQ thru the parameter generation feature.

The test currently has a problem in which the last row of the points dont vary the point size from the vertex export which is under review.Clipping is disabled until review.

Change 70883 on 2002/12/12 by csampayo@fl_csampayo2_r400

Adding new SU point test for size in vertex

Change 70838 on 2002/12/12 by georgev@devel_georgev_r400_lin2_marlboro

Added new tests.

Change 70753 on 2002/12/12 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 70742 on 2002/12/12 by mkelly@fl_mkelly_r400_win_laptop

ROM disable pixel pipes 2 and 3

Change 70739 on 2002/12/12 by mkelly@fl_mkelly_r400_win_laptop

ROM bad pipe disable 2 & 3 vertex pass

Change 70730 on 2002/12/12 by mkelly@fl_mkelly_r400_win_laptop

PA_SC_ENHANCE.PKR_ROW_WRAP_DISABLE test

Change 70726 on 2002/12/12 by ashishs@fl_ashishs_r400_win

updated comments and description. corrected cull radius in the test. generalised the test so that the point size can be entered using state registers in SU as well as vertex expot data using a single variable. (there doesn't seem to be problem with the texture as mentioned before because at low resolution its due to aliasing effect)

Change 70720 on 2002/12/12 by mkelly@fl_mkelly_r400_win_laptop

ROM disabled pipes 2 and 3

Change 70665 on 2002/12/12 by ashishs@fl_ashishs_r400_win

Sample test showing texture failure with vertex export point size. The texture breaks with point sizes of 8, 16 but passes with point size of 32. Also the texture breaks only on vertex export point size and not on the SU point size present. (the test has controls to enable/disable the vertex export point size)

Change 70567 on 2002/12/11 by viviana@viviana_crayola_linux_orl

Test to read the rbbm bus reads.

Change 70546 on 2002/12/11 by georgev@devel_georgev_r400_lin2_marlboro

Changed for different picture on output.

Change 70539 on 2002/12/11 by georgev@devel_georgev_r400_lin2_marlboro

Added new tests for predicated mova and sets.

Change 70463 on 2002/12/11 by ashishs@ashishs_crayola_linux_orl

Changed this file to run on linux. The test had problems with the Index buffer been written to memory multiple times unnecessarily.

Change 70450 on 2002/12/11 by vgoel@fl_vgoel2

added r400vgt_hos_tri_adaptive_complex test

Change 70448 on 2002/12/11 by vgoel@fl_vgoel2

modifed for higher tessellation level and also added texture mapping

Change 70416 on 2002/12/11 by vgoel@fl_vgoel2

changed the input texture file name from *.bmp to *.BMP

Change 70344 on 2002/12/11 by rramsey@RRAMSEY_P4_r400_win

fix problem with loading stipple repeat count value

Change 70329 on 2002/12/11 by omesh@omesh_r400_linux_marlboro_only_devel

Checked in a corner case for OnePixel8Fragments with all Red samples for
the one pixel.

Change 70306 on 2002/12/11 by rramsey@rramsey_crayola_linux_orl

Add randomization for perf countters

Change 70305 on 2002/12/11 by kevino@kevino_r400_win_marlboro

Added format comp cases to small prim, as well as 4x4, 8x8, and 16x16 (ARGB8888 only for now)

Change 70246 on 2002/12/10 by grayc@chip_regress_orl

fixed test name

Change 70236 on 2002/12/10 by markf@markf_tip_r400_linux_marlboro

Fixed Inifinity and NaN cases

Change 70208 on 2002/12/10 by markf@markf_tip_r400_linux_marlboro

Fixed to give infinity and NaN equal weight

Change 70169 on 2002/12/10 by georgev@devel_georgev_r400_lin2_marlboro

New tests added.

Change 70160 on 2002/12/10 by csampayo@fl_csampayo2_r400

Renable tests:
r400vgt_event_handling_03
r400vgt_event_handling_04

Add tests:
r400vgt_real_time_events_03
r400vgt_real_time_events_04

Change 70151 on 2002/12/10 by mkelly@fl_mkelly_r400_win_laptop

change to wait_gfx_idle_no_flush before read back of status regs

Change 70080 on 2002/12/10 by omesh@ma_omesh

Reverted back for the One Pixel cases to use seperate contexts for each fragment. I cannot change a register state within the same context, as there is no h/w synchronization featured for this.

Change 70053 on 2002/12/10 by omesh@omesh_r400_linux_marlboro_only_devel

Made the One Pixel cases use only a single context to simplify debugging
for Bill. The test always seemed to produce the 8 tiles it was supposed
to, at the RBT -> RBC interface. RC -> RB only sends a single
consolidated packed tile which is probably expanded into 8 tiles.

Change 69948 on 2002/12/10 by rramsey@rramsey_crayola_linux_orl

add usage comments

Change 69946 on 2002/12/10 by ctaylor@fl_ctaylor_r400_dtwin_marlboro

Update SC rand shell for new RTS trackers.

Change 69901 on 2002/12/10 by georgev@devel_georgev_r400_lin2_marlboro

Added simple move test.

Change 69878 on 2002/12/10 by ctaylor@fl_ctaylor_r400_dtwin_marlboro

Small update to SC rands
Fix Perf Counter PA bug with uninitialized select values.

Change 69834 on 2002/12/09 by georgev@devel_georgev_r400_lin2_marlboro

Added predicated kill tests.

Change 69833 on 2002/12/09 by csampayo@fl_csampayo_r400

Update for tiling

Change 69715 on 2002/12/09 by ctaylor@fl_ctaylor_r400_dtwin_marlboro

Add performance test

Change 69714 on 2002/12/09 by mkelly@fl_mkelly_r400_win_laptop

Fix print title...

Change 69709 on 2002/12/09 by omesh@omesh_r400_linux_marlboro_only_devel

Added 1 pixel testcases.

Change 69692 on 2002/12/09 by mkelly@fl_mkelly_r400_win_laptop

Enable clipper vertex reordering by default and update two test golds for regress_e to match...

Change 69668 on 2002/12/09 by ashishs@fl_ashishs_r400_win

initial checkin of perspective divide clipping tests with different primitive positions with respect to frustum

Change 69666 on 2002/12/09 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint on new test...

Change 69640 on 2002/12/09 by omesh@omesh_r400_linux_marlboro_only_devel

Made some changes to make sure the sample mask is set correctly in the
PA, to make sure I render only a single sample within a single pixel for
all 8 fragments. The waveforms don't show the right number of tile
probes coming down, neither does the course sample mask seem correct.

Change 69628 on 2002/12/09 by csampayo@fl_csampayo2_r400

Adjust image size for tiling

Change 69617 on 2002/12/09 by omesh@omesh_r400_linux_marlboro_only_devel

Added some simplified 1 Pixel testcases for Bill Lawless.

Change 69604 on 2002/12/09 by llefebvr@llefebvre_laptop_r400_emu

Backing out the change to the muladd that broke many regression tests.

Change 69570 on 2002/12/09 by mkelly@fl_mkelly_r400_win_laptop

Update test to dump vertex array for debugging...

Change 69566 on 2002/12/09 by mkelly@fl_mkelly_r400_win_laptop

Read viz query status registers using r400vgt_viz_query_01 as baseline.

Change 69422 on 2002/12/06 by markf@markf_tip_r400_linux_marlboro

Fixed

Change 69362 on 2002/12/06 by omesh@omesh_r400_linux_marlboro_only_devel

Added code to turn off multisampling for the resolve pass. This still didn't make the resolve produce any colors on the final image.

Change 69311 on 2002/12/06 by omesh@omesh_r400_linux_marlboro_only_devel

Increased coverage of the test, so that the output image contains an
easily distinguishable visual result.

Change 69294 on 2002/12/06 by omesh@omesh_r400_linux_marlboro_only_devel

Made some test fixes: Moved center of sampling area to top left of pixel (where I intended it to
be). Added some RB registers that were not programmed.

Change 69266 on 2002/12/06 by bbuchner@fl_bbuchner_r400_win

corrected instances where I/O signals were being used.

Change 69250 on 2002/12/06 by omesh@omesh_r400_linux_marlboro_only_devel

Added a second seperate surface to test that is meant for the resolved
result. I now dump both the multisampled as well as the resolved image.
The resolved image looks wrong. (All black)

Change 69158 on 2002/12/06 by llefebvr@llefebvre_laptop_r400_emu

More HW precision fixes for the interpolators. Now matches perfectly HW for regular cases. Still a problem with very large numbers (but they are not used in the regression yet).

Change 69153 on 2002/12/06 by lseiler@lseiler_r400_win_marlboro2

Fixes minor bugs in resolve code and test

Change 69080 on 2002/12/06 by markf@markf_tip_r400_linux_marlboro

Added more test cases

Change 69008 on 2002/12/05 by markf@markf_tip_r400_linux_marlboro

Made the verts of the rectangle more random

Change 68976 on 2002/12/05 by vgoel@fl_vgoel2

modified test for tracing bug

Change 68960 on 2002/12/05 by ashishs@fl_ashishs_r400_win

updated

Change 68926 on 2002/12/05 by markf@markf_tip_r400_linux_marlboro

Increased range of vertex positions from 32 to 64

Change 68906 on 2002/12/05 by vgoel@fl_vgoel2

modified this test to output one object for debugging purpose.

Change 68878 on 2002/12/05 by ashishs@fl_ashishs_r400_win

added tests:
r400cl_frustum_LR_TB_01
r400cl_edgeflags_05
r400cl_edgeflags_06
r400cl_edgeflags_07
r400cl_cull_only_ena_02
r400cl_cull_only_ena_03
r400vte_z_fmt_02
r400vte_z_fmt_03
r400vte_z_fmt_04

Change 68870 on 2002/12/05 by ashishs@fl_ashishs_r400_win

Features Tested:frustum clipping; LRTB clip cases
Test Purpose:simple LRTB clip cases; 9 primitives; each primitive has a different combination of clipping control bits set for vertex 2
Expected Results: The primitives generated are as follows:
1. No clipping
2. Clipped on left edge of viewing frustum
3. Clipped on right of viewing frustum
4. Clipped on top edge of viewing frustum
5. Clipped on top/left corner of viewing frustum
6. Clipped on top/right corner of viewing frustum
7. Clipped on bottom edge of viewing frustum
8. Clipped on bottom/left corner of viewing frustum
9. Clipped on bottom/right corner of viewing frustum

Change 68862 on 2002/12/05 by ashishs@fl_ashishs_r400_win

tests having primitve types as QUAD_LIST,QUAD_STRIP and POLYGON. Each test premuting various edgeflag combinations for the primitive edges. UCP Clipping enabled with 4 planes and primitve mapped with 8 texture maps.

Currently all these 3 tests have a problem in which the edgeflag setting overrides the polymode linefill algorithm and hence causing extra edges to show show up inside the primitive. (the extra edges are the edges common to triangles inside the primitives)

Change 68861 on 2002/12/05 by ashishs@fl_ashishs_r400_win

The test has a bug currently in which the edgeflag setting overrides the polymode linefill algorithm thereby showing up and extra edge for the POLYGON(common edges between the 2 triangles of a POLYGON)

Change 68860 on 2002/12/05 by georgev@devel_georgev_r400_lin2_marlboro

Added predicated exports.

Change 68844 on 2002/12/05 by csampayo@fl_csampayo2_r400

Updated linear surfaces defs

Change 68839 on 2002/12/05 by omesh@omesh_r400_linux_marlboro_only_devel

Added 2 testcases per file to test SWAP with 128 bit per pixel color, as
Mark told me that this combination showed a bug.

Change 68823 on 2002/12/05 by ashishs@fl_ashishs_r400_win

The test has a bug currently in which the edgeflag setting overrides the polymode linefill algorithm thereby showing up and extra edge for the QUAD_STRIP(common edge between the 2 triangles of a QUAD)

Change 68803 on 2002/12/05 by omesh@omesh_r400_linux_marlboro_only_devel

Fixed the opcode I was using in the pixel shader (It was wrong earlier,
used for regular texture fetch, not multisampled texture fetch).
However, now the Texture Pipe asserts, so I have informed Jocelyn about
this.

Change 68800 on 2002/12/05 by markf@markf_tip_r400_linux_marlboro

Added some more random test cases

Change 68787 on 2002/12/05 by ashishs@fl_ashishs_r400_win

The test has a bug currently in which the edgeflag setting overrides the polymode linefill algorithm thereby showing up and extra edge for the QUAD_LIST(common edge between the 2 triangles of a QUAD)

Change 68786 on 2002/12/05 by omesh@omesh_r400_linux_marlboro_only_devel

Fixed the FETCH_COLOR_FRAGMENTS field.

Change 68785 on 2002/12/05 by sallen@sallen_r400_lin_marlboro

ferret: look for context_done in reg stream to indicate context busy.

Change 68784 on 2002/12/05 by omesh@omesh_r400_linux_marlboro_only_devel

A first cut at the multisample texture fetch and resolve tests. DumpView
doesn't display the output image correctly. Its probably not ready for
the resolve yet. Will conform with Jocelyn.

Change 68768 on 2002/12/05 by mkelly@fl_mkelly_r400_win_laptop

Fix read back

Change 68742 on 2002/12/05 by markf@markf_tip_r400_linux_marlboro

Limited interpolant range to 0.0 to 1.0

Change 68688 on 2002/12/05 by markf@markf_r400_win_marlboro

Added simple pass thru shader for interpolation tests

Change 68615 on 2002/12/04 by omesh@omesh_r400_linux_marlboro_only_devel

Added 7 testcases to test RB fog blending in COLOR_8 mode.

Change 68612 on 2002/12/04 by markf@markf_r400_win_marlboro

Simple test for interpolators

Change 68577 on 2002/12/04 by abeaudin@abeaudin_r400_win_marlboro

fixed blender precision problem

Change 68542 on 2002/12/04 by omesh@omesh_r400_linux_marlboro_only_devel

Fixed Buzilla bug#886. Color Mask plane bits now replicate within 32 bit
word, based on number of bytes per pixel.

Change 68507 on 2002/12/04 by llefebvr@llefebvre_laptop_r400_emu

The emulator was off by 1 bit in the adder. Need to update golds... They should not missmatch by more than 1 LSB.

Change 68486 on 2002/12/04 by omesh@omesh_r400_linux_marlboro_only_devel

Fixed bugzilla bug#887, explicitly sending down alpha channel for all
vertices, so emulator and h/w don't mismatch. Haven't tested this fix
though.

Change 68425 on 2002/12/04 by mkelly@fl_mkelly_r400_win_laptop

Add RealDOT dword = 0 to both scripts, must manually set to 1 for test

Change 68424 on 2002/12/04 by ashishs@fl_ashishs_r400_win

updated to run on Unix

Change 68421 on 2002/12/04 by ashishs@fl_ashishs_r400_win

cleaned up unused code

Change 68420 on 2002/12/04 by llefebvr@llefebvre_laptop_r400_emu

This is a test error. Since the test is using C0 to increase the range of the addresses, when you modify PS_BASE you also modify which constant gets picked for range increase. You should have used K0 instead to regardless of PS_BASE the same constant is always picked. The test has been corrected to reflect this.

Change 68413 on 2002/12/04 by rramsey@RRAMSEY_P4_r400_win

Constrain stipple_repeat load so that the value is greater than the current count

Remove reset of stipple_state register after a packet with auto_reset = NEVER, since it is not needed any more due to the constraint on stipple_repeat loads

Add randomization for sc_one_quad and quad_order

Change 68380 on 2002/12/04 by markf@markf_r400_linux_marlboro

Fixed build error

Change 68368 on 2002/12/04 by ashishs@fl_ashishs_r400_win

updated to work on linux

Change 68367 on 2002/12/04 by jhoule@jhoule_r400_win_marlboro

Various updates:
- Supports loading of cube maps
- Support automatic ordering of mip levels (when no level is specified)
- Support for constants in pixel and vertex shaders (starting at 0 for each)
- Exception trapping
- Static allocation workaround (for old primlib issue related to RTTI)

Change 68365 on 2002/12/04 by viviana@viviana_crayola_linux_orl

Took out the writes and reads of the viz query status registers.

Change 68342 on 2002/12/04 by georgev@devel_georgev_r400_lin2_marlboro

Init loop counters in default.

Change 68273 on 2002/12/03 by ygiang@ygiang_r400_linux_marlboro

fixed: alu constant index

Change 68269 on 2002/12/03 by markf@markf_r400_win_marlboro

Simple test for fast color clear and fast color expand.

Change 68191 on 2002/12/03 by llefebvr@llefebvre_laptop_r400_emu

new implementation of the DOT product. Currently turned off by default. Need to set RealDOT to 1 in registry file to turn it on. Changed regular mulladd to reflect implementation changes as well.

Change 68173 on 2002/12/03 by ashishs@fl_ashishs_r400_win

To Test Culling when UCPs are enabled

Change 68154 on 2002/12/03 by omesh@omesh_r400_linux_marlboro_only_devel

Fixed bugzilla#881 to program the number format in RB correctly, based
on the color surface format used.

Change 68126 on 2002/12/03 by ashishs@fl_ashishs_r400_win

To check Z multiply 1/W using VTX_Z_FMT = 0 Texture included to assure integrity of parameter cache indices is maintained.

Change 68088 on 2002/12/03 by mkelly@fl_mkelly_r400_win_laptop

Remove SC hos test for the time being...

Change 68077 on 2002/12/03 by ashishs@fl_ashishs_r400_win

fixed texture related problem

Change 68073 on 2002/12/03 by rramsey@RRAMSEY_P4_r400_win

Enhance sc script
Can disable pa sims with 'nopa' cmd line arg
Can run with coverage on linux with 'cov' cmd line arg
Can pause regression by touching pause_regression in test directory

Change 67985 on 2002/12/02 by markf@markf_r400_win_marlboro

Added the test cases

Change 67969 on 2002/12/02 by ygiang@ygiang_r400_linux_marlboro

fixed: wasn't doing the right lit function before

Change 67962 on 2002/12/02 by vgoel@fl_vgoel2

added adaptive tri-patch test

Change 67960 on 2002/12/02 by vgoel@fl_vgoel2

added adaptive line patch and rect patch tests to test_list and modified these tests
to export to one DWORD for one index ptr.

Change 67932 on 2002/12/02 by ashishs@fl_ashishs_r400_win

updated since failing in tiled mode

Change 67910 on 2002/12/02 by mkelly@fl_mkelly_r400_win_laptop

Set nan_retain in CL to false...

Change 67896 on 2002/12/02 by vgoel@fl_vgoel2

added adaptive line patch tessellation test

Change 67881 on 2002/12/02 by georgev@devel_georgev_r400_lin2_marlboro

Initialized unused loops for loop_2 test.

Change 67857 on 2002/12/02 by ygiang@ygiang_r400_linux_marlboro

fixed: bugs in test cases

Change 67849 on 2002/12/02 by smoss@smoss_crayola_linux_orl_regress

modified for Linux

Change 67847 on 2002/12/02 by vgoel@fl_vgoel2

stored index pointers to rect patches in one dword

Change 67801 on 2002/12/02 by mkelly@fl_mkelly_r400_win_laptop

Set R400HardwareAccurate to default on in both scripts.
Set Tiling to default on in both scripts.

Change 67782 on 2002/12/02 by rramsey@RRAMSEY_P4_r400_win

temp fix to get rands passing vs tb_sc

Change 67779 on 2002/12/02 by ashishs@fl_ashishs_r400_win

updated

Change 67778 on 2002/12/02 by markf@markf_r400_lt_marlboro

Add source file

Change 67759 on 2002/12/02 by ashishs@fl_ashishs_r400_win

XY multiply 1/W using VTX_XY_FMT = 0 Texture included to assure integrity of parameter cache indices is maintained.

Change 67756 on 2002/12/01 by ashishs@fl_ashishs_r400_win

To check the W0 bit in the VTE_CNTL register. primitive mapped with 8 textures.

Change 67748 on 2002/12/01 by ashishs@fl_ashishs_r400_win

vte test : To check the W0 bit in the VTE_CNTL register

Change 67548 on 2002/11/29 by smoss@smoss_crayola_linux_orl_regress

increased timeout

Change 67318 on 2002/11/27 by ygiang@ygiang_r400_linux_marlboro

added:more mova tests

Change 67302 on 2002/11/27 by ygiang@ygiang_r400_linux_marlboro

more sp tests

Change 67285 on 2002/11/27 by ygiang@ygiang_r400_linux_marlboro

fixed: test cases

Change 67277 on 2002/11/27 by georgev@devel_georgev_r400_lin2_marlboro

Added and fixed tests.

Change 67273 on 2002/11/27 by mkelly@fl_mkelly_r400_win_laptop

Verify SU_SC_MODE persp corr disable

Change 67256 on 2002/11/27 by smoss@smoss_crayola_linux_orl

edit for Linux

Change 67255 on 2002/11/27 by mkelly@fl_mkelly_r400_win_laptop

Update test to pass at GC level, including alloc TILE memory and setting CMASK

Change 67254 on 2002/11/27 by jhoule@jhoule_r400_win_marlboro

Updated SET_TEX_LOD code generator for new syntax.

Change 67244 on 2002/11/27 by mkelly@fl_mkelly_r400_win_laptop

Update..

Change 67170 on 2002/11/27 by hwise@fl_hwise_r400_win

Test clean up

Change 67133 on 2002/11/26 by smoss@smoss_crayola_win

SU TESTS

Change 67131 on 2002/11/26 by ygiang@ygiang_r400_linux_marlboro

added: sp mova tests

Change 67101 on 2002/11/26 by ashishs@fl_ashishs_r400_win

This is a sample test to verify the barycentric coordinates. The test generates a color band texture of 8 colors with each color varying from the lowest intensity to the highest intensity(0-255). The test uses display size of 256X256 so that each row in the frameBufferDump has a different intensity of color for each color band. The primitve is then clipped with guard bands and 6 clipping planes and the result is verified with unclipped image(same test with clip disable)

(If the HW accurate mode isnt on then the image will have the center rows 128 and 129 with the same color intensities but the problem disappears when HW accurate mode is turned on)

Change 67081 on 2002/11/26 by vgoel@fl_vgoel2

modified to include prim_order and pixel shader is modified

Change 67061 on 2002/11/26 by ashishs@fl_ashishs_r400_win

updated

Change 67056 on 2002/11/26 by georgev@devel_georgev_r400_lin2_marlboro

Ooops. Made negative.

Change 67055 on 2002/11/26 by georgev@devel_georgev_r400_lin2_marlboro

Added new tests.

Change 66906 on 2002/11/26 by omesh@omesh_r400_linux_marlboro_only_devel

Increased emulator timeout some more, as 2 of the testcases (all and fog_random failed last night's regression due to the timeout)

Change 66905 on 2002/11/26 by omesh@omesh_r400_linux_marlboro_only_devel

Changed the other tests that used the memory fill function to do a seperate allocate and load for all sets of vertex data.

Change 66878 on 2002/11/25 by ashishs@fl_ashishs_r400_win

This is a sample test to verify the perspective divide feature of the VTE along with clipping. Primitve Type : TRIANGLE LIST with 2 triangles forming a rectangle texture mapped with 8 textures. Clipping enabled with horizonatl and vertical guard bands and 6 ucp clip planes.

(found differences in clipped and unclipped images. under review)

Change 66869 on 2002/11/25 by csampayo@fl_csampayo2_r400

Adjust non-RTS rect and RTS rect sizes

Change 66868 on 2002/11/25 by csampayo@fl_csampayo2_r400

Added new RTS test

Change 66841 on 2002/11/25 by georgev@devel_georgev_r400_lin2_marlboro

New tests added.

Change 66797 on 2002/11/25 by georgev@devel_georgev_r400_lin2_marlboro

Added new tests.

Change 66771 on 2002/11/25 by vgoel@fl_vgoel2

added prim_order and correct few cpp files.

Change 66770 on 2002/11/25 by omesh@omesh_r400_linux_marlboro_only_devel

Added load vertex code within the render loop to allocate different sections of memory instead of reusing the same section. (This does not require the wait_gfx_idle() call to synchronize between various passes of memory writes). If this works, I will change all the other files for John Chen.

Change 66727 on 2002/11/25 by viviana@viviana_crayola_linux_orl

New test to test the reads from the rbbm - sc block.

Change 66683 on 2002/11/25 by mkelly@fl_mkelly_r400_win_laptop

More SC HW coords testing...
Fix regress_r400 to unzip framebuf with option z

Change 66678 on 2002/11/25 by vgoel@fl_vgoel2

added test r400vgt_hos_PNT_adaptive_complex to test_list
changed r400vgt_hos_PNQ_adaptive_complex.cpp to include prim_order

Change 66606 on 2002/11/25 by ashishs@fl_ashishs_r400_win

updated comments

Change 66584 on 2002/11/25 by mkelly@fl_mkelly_r400_win_laptop

SC HW coords extremity testing...

Change 66576 on 2002/11/25 by mkelly@fl_mkelly_r400_win_laptop

Barycentric test on SC HW coords extremes

Change 66543 on 2002/11/24 by ygiang@ygiang_r400_linux_marlboro

added: fog and other sp tests

Change 66537 on 2002/11/24 by mmantor@mmantor_r400_win

increased timeout_max in test for Pipe disable testing

Change 66505 on 2002/11/23 by smoss@smoss_crayola_linux_orl_regress

<Orlando Hardware Regression Results >

Change 66401 on 2002/11/22 by ygiang@ygiang_r400_win_marlboro_p4

changes for hardware accurate mode

Change 66374 on 2002/11/22 by ygiang@ygiang_r400_linux_marlboro

added: tests with new names

Change 66373 on 2002/11/22 by ygiang@ygiang_r400_linux_marlboro

deleted: old test name

Change 66372 on 2002/11/22 by georgev@devel_georgev_r400_lin2_marlboro

Added new tests.

Change 66367 on 2002/11/22 by csampayo@fl_csampayo_r400

Updated test list and test tracker for the following tests:
r400vgt_real_time_events_01
r400vgt_real_time_events_02

Change 66361 on 2002/11/22 by csampayo@fl_csampayo2_r400

Add new RTS test

Change 66360 on 2002/11/22 by csampayo@fl_csampayo2_r400

Fix initialization of context 0 and general cleanup

Change 66343 on 2002/11/22 by ygiang@ygiang_r400_linux_marlboro

added: shader mova test

Change 66300 on 2002/11/22 by omesh@omesh_r400_linux_marlboro_only_devel

Removed the wait_gfx_idle() call within loops. There's now just one at the end of the test. I must have overlooked these tests when I did the others. Also compiled them to check that they all compile.

Change 66265 on 2002/11/22 by mkelly@fl_mkelly_r400_win_laptop

Test baryc interpolation of a single line -4k to (12k-1) range in X major

Change 66169 on 2002/11/22 by mkelly@fl_mkelly_r400_win_laptop

Only unzip a framebuf file if that file is to be compared...

Change 66111 on 2002/11/21 by ashishs@fl_ashishs_r400_win

updated

Change 66093 on 2002/11/21 by ygiang@ygiang_r400_linux_marlboro

fixed: alu constant reg in test

Change 66025 on 2002/11/21 by georgev@devel_georgev_r400_lin2_marlboro

Added new tests.

Change 66008 on 2002/11/21 by omesh@omesh_r400_linux_marlboro_only_devel

Added tile buffer initialized versions of the Depth tests that use
Larry's tile buffer data word initialization routines from addrenum(.h
and .c)

Change 65995 on 2002/11/21 by smoss@smoss_crayola_linux_orl_emu_regress

increased timeout

Change 65994 on 2002/11/21 by rramsey@RRAMSEY_P4_r400_win

tb_sc.v - Put in new method for context management to fix failing vizq test
Add DISABLE_RBBM_FILTER parameter so we can test with all rbbm data flowing

rand_r400sc.sh - Add some commenting to rtl sim report files

Change 65987 on 2002/11/21 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 65973 on 2002/11/21 by llefebvr@llefebvr_r400_linux_marlboro

fixing constant setup bug in the test

Change 65880 on 2002/11/21 by georgev@devel_georgev_r400_lin2_marlboro

Added ten_loops.

Change 65854 on 2002/11/21 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 65847 on 2002/11/21 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 65842 on 2002/11/21 by ashishs@fl_ashishs_r400_win

removed the dump file compares

Change 65832 on 2002/11/21 by mkelly@fl_mkelly_r400_win_laptop

Fix agp_r handling cause I broke it with previous check-in :)

Change 65830 on 2002/11/21 by georgev@devel_georgev_r400_lin2_marlboro

Added more loop tests.

Change 65828 on 2002/11/21 by mkelly@fl_mkelly_r400_win_laptop

* Added option -w, where the user can define the $src_path, this will
  allow regress_r400 script to run when Perforce is down. The user
  needs to hardcode his $src_path in regress_r400. Search on variable
  $use_default_root and edit $src_path in the else clause.

* Fixed bug when reading in multiple compare arguments in test_list

Change 65827 on 2002/11/21 by georgev@devel_georgev_r400_lin2_marlboro

Buttoned up test because we're not using it anymore.

Change 65768 on 2002/11/20 by ygiang@ygiang_r400_linux_marlboro

added: mova w/ max alu constant regs use

Change 65751 on 2002/11/20 by vgoel@fl_vgoel2

changed it back to displaying 28 objects

Change 65745 on 2002/11/20 by jhoule@jhoule_r400_win_marlboro

XPLAT format of variable declaration.
Cleanup of variables and flags; still some left, but should be easier now.

Flag -v only happens under UNIX.

Change 65630 on 2002/11/20 by mkelly@fl_mkelly_r400_win_laptop

Fix uninitialized W...

Change 65610 on 2002/11/20 by ygiang@ygiang_r400_linux_marlboro

fixed: triangle sizes for debug

Change 65554 on 2002/11/20 by abeaudin@abeaudin_r400_win_marlboro

regression files

Change 65538 on 2002/11/20 by abeaudin@abeaudin_r400_win_marlboro

adding rb tests

Change 65533 on 2002/11/20 by vgoel@fl_vgoel2

changed test temporarily to output one line curve

Change 65527 on 2002/11/20 by ashishs@fl_ashishs_r400_win

updated to compare frameBufferDump, VapV300, ClipGa_alg and SuScan

Change 65511 on 2002/11/20 by mkelly@fl_mkelly_r400_win_laptop

add another tricky poly offset quick test to regress_e...

Change 65509 on 2002/11/20 by ashishs@fl_ashishs_r400_win

changed frameBufferDump in multiples of 32

Change 65503 on 2002/11/20 by ashishs@fl_ashishs_r400_win

updated the frameBufferDump in multiples of 32

Change 65487 on 2002/11/20 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 65481 on 2002/11/20 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 65477 on 2002/11/20 by smoss@smoss_crayola_linux_orl_regress

added rom_sc.dmp

Change 65420 on 2002/11/19 by ashishs@fl_ashishs_r400_win

updated for texture related problem.

for each texture added

point_texture_constant.setMIP_FILTER(TFetchConst::Mip_BaseMap);

Change 65410 on 2002/11/19 by ashishs@fl_ashishs_r400_win

ucp clipping of triangle strip mapped with 8 textures with edgeflag combinations

Change 65395 on 2002/11/19 by ashishs@fl_ashishs_r400_win

edgeflag clipping with 8 textures. (primitive types: Triangle with WFLAGS & TRIANGLE FAN)

Change 65369 on 2002/11/19 by georgev@devel_georgev_r400_lin2_marlboro

Consolidated SQ tests.

Change 65367 on 2002/11/19 by llefebvr@llefebvre_laptop_r400_emu

This wasn't hung. Just took a lot of time. I reduced the loop count to something more reasonable. Also changed the R400_NAN to FFC00000.

Change 65318 on 2002/11/19 by omesh@ma_omesh

Added an explicit static cast for a conversion from int->enum. These tests should also now compile on Windows, as I verified.

Change 65282 on 2002/11/19 by vgoel@fl_vgoel2

added r400vgt_hos_pnl_adaptive_complex to gold

Change 65279 on 2002/11/19 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 65249 on 2002/11/19 by mkelly@fl_mkelly_r400_win_laptop

Test which shows an interpolator bug...

Change 65239 on 2002/11/19 by csampayo@fl_csampayo2_r400

Updated for proper timing between packets

Change 65234 on 2002/11/19 by csampayo@fl_csampayo_r400

Added 3 new VGT tests and updated test_list and test tracker accordingly

Change 65219 on 2002/11/19 by mkelly@fl_mkelly_r400_win_laptop

regress_e face bit check...

Change 65199 on 2002/11/19 by llefebvr@llefebvre_laptop_r400_emu

There were two problems with this test:
1) The first pass pixel shader wasn't exporting anything thus the SX wasn't sending anything to the RBs hence the hang. You should never have a dummy pixel shader if there are pixel generated.
2) The time allowed to the test was too short. I incremented it to 10000 (was 1000).

I took the opportunity to do some code cleanup as well. Rerun other memory export tests to make sure I did not break anything.

Change 65189 on 2002/11/19 by mkelly@fl_mkelly_r400_win_laptop

Regress_e test to check XY position with combinations of CENTERs, CENTROIDs

Change 65181 on 2002/11/19 by mkelly@fl_mkelly_r400_win_laptop

Short test for regress_e...

Change 65180 on 2002/11/19 by mkelly@fl_mkelly_r400_win_laptop

Fix cull direction now that face bit detection is shader is working properly...

Change 65168 on 2002/11/19 by mkelly@fl_mkelly_r400_win_laptop

Fix cull direction to CW, now passes with interpolators face bit fix, change 64957.

Change 65161 on 2002/11/19 by ashishs@fl_ashishs_r400_win

cleaned up code

Change 65056 on 2002/11/18 by ashishs@fl_ashishs_r400_win

redisgned this test since failing in tiled mode. Now the test passes the old linear dump as well as the Tiled dump compare.

Change 65043 on 2002/11/18 by ygiang@ygiang_r400_linux_marlboro

fixed: plgx time out for large triangle

Change 65006 on 2002/11/18 by omesh@omesh_r400_linux_marlboro_only_devel

Increased default emulator timeout of tests (800), to ensure they don't
timeout during nightly regressions. Don't know why the same tests take
factors of more emulator cycles.

Change 65002 on 2002/11/18 by georgev@devel_georgev_r400_lin2_marlboro

Changes to make

Change 64926 on 2002/11/18 by vgoel@fl_vgoel2

added vgt_grp_prim_order type

Change 64906 on 2002/11/18 by ygiang@ygiang_r400_linux_marlboro

modified: output channels

Change 64870 on 2002/11/18 by mkelly@fl_mkelly_r400_win_laptop

Fix for 2^23 bias...

Change 64863 on 2002/11/18 by csampayo@fl_csampayo_lt_r400

Temporarily comment out tests:
r400vgt_event_handling_03

r400vgt_event_handling_04

Change 64858 on 2002/11/18 by mkelly@fl_mkelly_r400_win_laptop

Update same changelist 61027, 2^23 bias

Change 64848 on 2002/11/18 by ashishs@fl_ashishs_r400_win

updated to change the framebufferDump size

Change 64837 on 2002/11/18 by mkelly@fl_mkelly_r400_win_laptop

Add negative window offset to regress_e...

Change 64829 on 2002/11/18 by ashishs@fl_ashishs_r400_win

added triangles with edgeflags and texture test to the tracker

Change 64795 on 2002/11/18 by mkelly@fl_mkelly_r400_win_laptop

* sq_block_model.cpp was not adding the buffer offset for centriods when centers and centriods are sent.
* fixed gold for r400sc_msaa_8_primtypes_01
* add r400sc_sp_sample_cntl_01 to SC regress_e to lock in this test and help minimize future debugging efforts.

Change 64760 on 2002/11/18 by mkelly@fl_mkelly_r400_win_laptop

Correct golds....

Change 64652 on 2002/11/16 by csampayo@fl_csampayo2_r400

Added wait without flush between packets

Change 64564 on 2002/11/15 by ashishs@fl_ashishs_r400_win

tests updated (vertex buffer not set properly causing tests to FAIL, also added render_engine.Wait_Gfx_Idle_No_Flush();)

Change 64563 on 2002/11/15 by ygiang@ygiang_r400_linux_marlboro

added: absolute alu constant registers test

Change 64562 on 2002/11/15 by ashishs@fl_ashishs_r400_win

tests updated (vertex buffer not set properly causing tests to FAIL, also added render_engine.Wait_Gfx_Idle_No_Flush();)

Change 64555 on 2002/11/15 by omesh@omesh_r400_linux_marlboro_only_devel

Added the 7 test names for which I have generated golden images, to be
used with the "make regress_e" target. Will add more later.

Change 64542 on 2002/11/15 by omesh@omesh_r400_linux_marlboro_only_devel

Checking in golden result images for 7 testcases which I can predict the image of (offhand). The other
testcases, I would have to spend more time to visually verify. Also checking in the magic golden.lst
file that supposedly keeps track of tests marked as having golden results.

Change 64525 on 2002/11/15 by mkelly@fl_mkelly_r400_win_laptop

Golds for Frank Hsien's regression...

Change 64504 on 2002/11/15 by jhoule@jhoule_r400_win_marlboro

Updated shader programs to new SET_* syntax.

Change 64501 on 2002/11/15 by omesh@omesh_r400_linux_marlboro_only_devel

Added "standard" testcases for tests that we want to include in the RB mini regression. These happen to
correspond to some bugs testcases which Alicia is probably working on. For all future golden images, we will
wait on trying to freeze the emulator to be hardware accurate to 12 bits of blending and also wait on Paul's
make target that would support testcase based golden file creation / regression.

Change 64455 on 2002/11/15 by kevino@kevino_r400_win_marlboro

test case files for 32as8 and 32as88

Change 64425 on 2002/11/15 by abeaudin@abeaudin_r400_win_marlboro

added fog test

Change 64399 on 2002/11/15 by ashishs@fl_ashishs_r400_win

updated for texture related problem.

for each texture added

point_texture_constant.setMIP_FILTER(TFetchConst::Mip_BaseMap);

Change 64383 on 2002/11/15 by omesh@omesh_r400_linux_marlboro_only_devel

Increased emulator timeout for these tests. For some reason they worked earlier, but some change in the emulator might have caused
these tests to take more "emulator cycles" than before. Verified that these tests don't timeout assert anymore, so they should pass
correctly with the next nightly regression.

Change 64360 on 2002/11/15 by jhoule@jhoule_r400_win_marlboro

Modified test program to prevent sending TILED 1D textures.

Change 64354 on 2002/11/15 by ygiang@ygiang_r400_linux_marlboro

fixed: for debug

Change 64310 on 2002/11/15 by rramsey@rramsey_crayola_linux_orl

shell script for running emulator sc rands (works on linux and win)
runs pa and sc testbenches

Change 64284 on 2002/11/15 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 64207 on 2002/11/14 by vgoel@fl_vgoel2

added r400vgt_hos_PNQ_adaptive_complex

Change 64205 on 2002/11/14 by vgoel@fl_vgoel2

modified reuse number and level of tessellation

Change 64200 on 2002/11/14 by ygiang@ygiang_r400_linux_marlboro

added: pixel cube test

Change 64162 on 2002/11/14 by vgoel@fl_vgoel2

added wait cycles after first pass

Change 64148 on 2002/11/14 by rramsey@RRAMSEY_P4_r400_win

move flush back to end of test, add perfcounter setup/reads

Change 64125 on 2002/11/14 by ashishs@fl_ashishs_r400_win

This test is intended to validate the clippper processing of the edge flags for the triangle list primitive type.
Clipping with 4 UCPs is enabled and actual clipping takes place
The edge flags for each primitive are permuted accross all packets

Each primitive has position, 2 colors and 8 textures.

Change 64118 on 2002/11/14 by vgoel@fl_vgoel2

updated for not writing unknown value to export memory. Also updated is gold.

Change 64086 on 2002/11/14 by vgoel@fl_vgoel2

deleted these files

Change 64082 on 2002/11/14 by vgoel@fl_vgoel2

changed r400_local_tonemapping to r400vgt_local_tonemapping

Change 64068 on 2002/11/14 by mkelly@fl_mkelly_r400_win_laptop

Add if statement to avoid annoying message about cannot unzip an agp_r file...

Change 64060 on 2002/11/14 by georgev@devel_georgev_r400_lin2_marlboro

Fix bad regressions and put in tests for yung to use.

Change 64024 on 2002/11/14 by ygiang@ygiang_r400_linux_marlboro

fixed: else statement for random test case

Change 64023 on 2002/11/14 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 64022 on 2002/11/14 by mkelly@fl_mkelly_r400_win_laptop

Add textured line to regress_e

Change 64017 on 2002/11/14 by mkelly@fl_mkelly_r400_win_laptop

Update test to behave identically to syncs prior to 60942

Change 63973 on 2002/11/14 by kevino@kevino_r400_win_marlboro

Removes non-existant fmt_32_as_8_float and fmt_32_as_8_8_float cases.

Change 63954 on 2002/11/14 by kevino@kevino_r400_win_marlboro

Was missing a semi-colon at the end of some of the muladd commands in the created .sp files.

Change 63950 on 2002/11/14 by kevino@kevino_r400_win_marlboro

fixed pitch select for texture dump to be min of 32.

Change 63928 on 2002/11/14 by mkelly@fl_mkelly_r400_win_laptop

Change gold path back to gold and not gold_tile_enabled!

Change 63927 on 2002/11/14 by mkelly@fl_mkelly_r400_win_laptop

Support for *.agp_r management and compare....

Change 63862 on 2002/11/13 by rmanapat@rmanapat_r400_sun_marlboro

Updated test so that ATI_MEMPAK error does not occur

Change 63859 on 2002/11/13 by csampayo@fl_csampayo2_r400

Added wait between passes to allow memory export to complete

Change 63852 on 2002/11/13 by ygiang@ygiang_r400_linux_marlboro

fixed:unknown value for vector

Change 63846 on 2002/11/13 by csampayo@fl_csampayo2_r400

Updates for SQ counter update

Change 63783 on 2002/11/13 by ashishs@fl_ashishs_r400_win

sample texture tests for customization

Change 63768 on 2002/11/13 by kevino@kevino_r400_win_marlboro

Added static render state because primlib can't seem to deal with dynamic ones all of a sudden.

Change 63750 on 2002/11/13 by kevino@kevino_r400_win_marlboro

Fix pitch to min 32

Change 63739 on 2002/11/13 by grayc@chip_regress_orl

golds for chip sims

Change 63733 on 2002/11/13 by mkelly@fl_mkelly_r400_win_laptop

Support <test_name>.agp_r in regress_r400

Change 63729 on 2002/11/13 by grayc@chip_regress_orl

gold

Change 63728 on 2002/11/13 by grayc@chip_regress_orl

golds for chip sims

Change 63698 on 2002/11/13 by grayc@chip_regress_orl

gold checkins

Change 63660 on 2002/11/13 by llefebvr@llefebvre_laptop_r400_emu

New non constant fog test.

Change 63616 on 2002/11/13 by omesh@omesh_r400_linux_marlboro_only_devel

Checked in a temporary test I am working on. Will rename later.

Change 63602 on 2002/11/13 by rramsey@RRAMSEY_P4_r400_win

change rands so we only wait_gfx_idle once per case

Change 63599 on 2002/11/13 by mkelly@fl_mkelly_r400_win_laptop

To debug color change issue...

Change 63595 on 2002/11/13 by mkelly@fl_mkelly_r400_win_laptop

Uncomment test...

Change 63588 on 2002/11/13 by mkelly@fl_mkelly_r400_win_laptop

Comment out aborting test for the time being...

Change 63552 on 2002/11/12 by csampayo@fl_csampayo_lt_r400

Updated test_list and test tracker for the following tests:
r400vgt_multi_pass_pix_shader_07
r400vgt_multi_pass_pix_shader_08

Change 63547 on 2002/11/12 by csampayo@fl_csampayo_r400

Adding 2 new VGT multi-pass pixel shader tests with quad_order_enable bit on.

Change 63466 on 2002/11/12 by georgev@devel_georgev_r400_lin2_marlboro

Changed comments, verify infinity works.

Change 63464 on 2002/11/12 by ashishs@fl_ashishs_r400_win

sorted test_list

Change 63458 on 2002/11/12 by ashishs@fl_ashishs_r400_win

update

Change 63441 on 2002/11/12 by llefebvr@llefebvre_laptop_r400_emu

Adding mova stress tests for the SQ.

Change 63435 on 2002/11/12 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 63359 on 2002/11/12 by mkelly@fl_mkelly_r400_win_laptop

Switch to tiling...

Change 63349 on 2002/11/12 by kevino@kevino_r400_linux_marlboro

Needed for smallprim tests

Change 63333 on 2002/11/12 by smoss@smoss_crayola_win

update golds

Change 63325 on 2002/11/12 by mkelly@fl_mkelly_r400_win_laptop

Modify for tiling...

Change 63313 on 2002/11/12 by omesh@omesh_r400_linux_marlboro_only_devel

Added the color_2_10_10_10_number_gamma as upto 12 bit gamma is now supported.

Change 63271 on 2002/11/11 by csampayo@fl_csampayo_r400

Added explicit initialization of the quad_order_enable bit in PA_SU_SC_MODE_CNTL register

Change 63233 on 2002/11/11 by ashishs@fl_ashishs_r400_win

edited comment

Change 63232 on 2002/11/11 by ashishs@fl_ashishs_r400_win

Tests guard band clipping(TRIANGLE_WITH_WFLAGS)
6 primitives in each of 4 quadrants.
Top Left Quadrant, gouraud shading

Top Right Quadrant, FLAT SHADING with START vertex as provoking vtx
Lower Right Quadrant, FLAT SHADING with END vertex as provoking vtx
Lower Left Quadrant, SIX TEXTURES, with COLOR0 as transparent
Expected Results: 6 primitives in each of four quadrants, guard band clipped

Change 63229 on 2002/11/11 by georgev@devel_georgev_r400_lin2_marlboro

Added more random cases.

Change 63217 on 2002/11/11 by llefebvr@llefebvre_laptop_r400_emu

Fixing an error with loops in the SQ that caused it to loop indefinitely. Also adding some more predication tests.

Change 63182 on 2002/11/11 by mkelly@fl_mkelly_r400_win_laptop

MSAA Primtypes SC for regress_e

Change 63180 on 2002/11/11 by mkelly@fl_mkelly_r400_win_laptop

Multi-textures, big vertex for regress_e

Change 63173 on 2002/11/11 by mkelly@fl_mkelly_r400_win_laptop

Add test to regress_e, SC sample control XY positions

Change 63168 on 2002/11/11 by smoss@smoss_crayola_win

update golds

Change 63143 on 2002/11/11 by ctaylor@fl_ctaylor_r400_dtwin_marlboro

Changed rands to insert wait_gfx_idle between each packet to get around (temporarily) RBBM issues.

Change 63113 on 2002/11/11 by llefebvr@llefebvre_laptop_r400_emu

Submiting an example of how to use the SQ generated counters.

Change 63109 on 2002/11/11 by georgev@devel_georgev_r400_lin2_marlboro

Fix seg fault.

Change 63101 on 2002/11/11 by hwise@fl_hwise_r400_win

Removed blank sapce at end of line which broke the:
export REGRESS_DIFF_OPTIONS = IGNORE_DUMP_HEADERS

Change 63100 on 2002/11/11 by ashishs@fl_ashishs_r400_win

Tests guard band clipping(POLYGON)
4 primitives in 4 quadrants.
Top Left Quadrant, gouraud shading
Top Right Quadrant, FLAT SHADING with START vertex as provoking vtx
Lower Right Quadrant, FLAT SHADING with END vertex as provoking vtx
Lower Left Quadrant, SIX TEXTURES, with COLOR0 as transparent

Expected Results: 1 polygon in each of four quadrants, guard band clipped

Change 63071 on 2002/11/11 by smoss@smoss_crayola_win

allowed vte to do 1/w rather than su

Change 62910 on 2002/11/08 by csampayo@fl_csampayo_lt_r400

Updated status in test tracker and added to test_list: r400vgt_real_time_events_01

Change 62841 on 2002/11/08 by omesh@omesh_r400_linux_marlboro_only_devel

Added another series of 4 multibuffer golden images to test if the golden release process works for multibuffer dump testcases.

Change 62832 on 2002/11/08 by omesh@omesh_r400_linux_marlboro_only_devel

Added a first golden image to get familiar with the process and also verify that the golden image release process is working.

Change 62817 on 2002/11/08 by omesh@omesh_r400_linux_marlboro_only_devel

Added empty list of images marked as "golden" to start using Paul's gold.pl script to start checking in golden images.

Change 62791 on 2002/11/08 by georgev@devel_georgev_r400_lin2_marlboro

Test mova including rounding.

Change 62783 on 2002/11/08 by csampayo@fl_csampayo_r400

Update to properly init RTS context

Change 62780 on 2002/11/08 by subad@subad_r400_win_marlboro

Updated

Change 62776 on 2002/11/08 by mkelly@fl_mkelly_r400_win_laptop

Add two new registery settings for bad pipe and SC packer optimize

Change 62748 on 2002/11/08 by ashishs@fl_ashishs_r400_win

update

Change 62746 on 2002/11/08 by ashishs@fl_ashishs_r400_win

adding more frustum clipping tests

Change 62733 on 2002/11/08 by vgoel@fl_vgoel2

corrected this test

Change 62731 on 2002/11/08 by csampayo@fl_csampayo2_r400

Update to RTS data

Change 62711 on 2002/11/08 by ashishs@fl_ashishs_r400_win

updated for last 3 vte tests viz r400vte_pos_neg_combos_01/02/03

Change 62687 on 2002/11/08 by kevino@kevino_r400_win_marlboro

Added texture dumps (ppm and framebuf)

Change 62588 on 2002/11/07 by csampayo@fl_csampayo2_r400

Initial check in of first RTS test

Change 62542 on 2002/11/07 by omesh@omesh_r400_linux_marlboro_only_devel

Added 11 more extensive fog with color blending tests. These tests are
the same as the plain fog tests, except that they render half the color
saturation value as before and write them to the destination. In the
second pass, they re-render the same triangles and combine the earlier
half saturation from destination using color blending.

Change 62534 on 2002/11/07 by vgoel@fl_vgoel2

added rect adaptive tessellation test : still problem with tessellation factors

Change 62508 on 2002/11/07 by georgev@devel_georgev_r400_lin2_marlboro

Added mul_prev2 tests.

Change 62486 on 2002/11/07 by llefebvr@llefebvre_laptop_r400_emu

New SQ predication tests.

Change 62456 on 2002/11/07 by mkelly@fl_mkelly_r400_win_laptop

Switch from linear to tiling...

Change 62455 on 2002/11/07 by omesh@omesh_r400_linux_marlboro_only_devel

Added 11 more extensive fog tests. The emulator results of the random testcase
(fog_random) look strange and are possibly incorrect. I'll wait for the
first regression run before I file any bug. May add more testcases to
this file later this evening or tommorow.

Change 62419 on 2002/11/07 by mkelly@fl_mkelly_r400_win_laptop

Update for tiling...

Change 62411 on 2002/11/07 by grayc@chip_regress_orl

new test lists

Change 62398 on 2002/11/07 by mkelly@fl_mkelly_r400_win_laptop

Update for tiling...

Change 62315 on 2002/11/07 by mkelly@fl_mkelly_r400_win_laptop

Update to work with COLOR::SURFACE class and tiling mode

Change 62290 on 2002/11/07 by rramsey@RRAMSEY_P4_r400_win

add to depot

Change 62276 on 2002/11/07 by georgev@devel_georgev_r400_lin2_marlboro

Added mova_floor tests to list.

Change 62258 on 2002/11/07 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 62238 on 2002/11/06 by ashishs@fl_ashishs_r400_win

to show nan kill for points

Change 62168 on 2002/11/06 by georgev@devel_georgev_r400_lin2_marlboro

New tests for mova_floor.

Change 62144 on 2002/11/06 by vgoel@fl_vgoel2

added compex adaptive tessellation test for PNQ and added some more cool images

Change 62143 on 2002/11/06 by vgoel@fl_vgoel2

modified test for exporting to off-screen fram buffer

Change 62116 on 2002/11/06 by omesh@omesh_r400_linux_marlboro_only_devel

Fixed a bug related to the 2nd pass of blending over the same triangles,
related to bringing out the effects of the Alpha channel using Color.
Also switched off the "MAKE_VISUAL" option used to bring out Alpha using
Color. This will reduce the number of render states sent to the CP ring
buffer and hopefully be within the limit of 7 render state sets.
Please note that the h/w interface dump files will change and so will
the Color channel of the Framebuffer. The Alpha channel of the
framebuffer should remain unchanged.

Change 62103 on 2002/11/06 by ashishs@fl_ashishs_r400_win

VTE test: prim type- Point, cycling Z scale and Z offset through all bits including sign changes

Change 62079 on 2002/11/06 by ashishs@fl_ashishs_r400_win

VTE test: prim type- Point, cycling Y scale and Y offset through all bits including sign changes

Change 62056 on 2002/11/06 by mkelly@fl_mkelly_r400_win_laptop

Make frame buffer dump multiple of 32 in DISP_X_DIM and DISP_Y_DIM for tiling

Change 62050 on 2002/11/06 by ashishs@fl_ashishs_r400_win

VTE test: prim type- Point, cycling X scale and X offset through all bits including sign changes

Change 62012 on 2002/11/06 by kevino@kevino_r400_linux_marlboro

Added test cases for simple2D_2x2 with a bunch of formats

Change 61904 on 2002/11/06 by mkelly@fl_mkelly_r400_win_laptop

Change dump boudaries to multiple of 32 so we can switch to tiling.

Change 61749 on 2002/11/05 by georgev@devel_georgev_r400_lin2_marlboro

Added a NULL check to see if the seg fault is fixed.

Change 61716 on 2002/11/05 by mkelly@fl_mkelly_r400_win_laptop

Comment out texture fills to the tiled framebuffer...

Change 61715 on 2002/11/05 by mkelly@fl_mkelly_r400_win_laptop

Comment out filling framebuffer with texture data to look correct in tiled mode..

Change 61656 on 2002/11/05 by mkelly@fl_mkelly_r400_win_laptop

Possible SQ interpolation bug...

Change 61492 on 2002/11/04 by georgev@devel_georgev_r400_lin2_marlboro

Test using the new register sets.

Change 61483 on 2002/11/04 by ygiang@ygiang_r400_linux_marlboro

added: co-issue and direct scalar export test

Change 61403 on 2002/11/04 by ashishs@fl_ashishs_r400_win

updated

Change 61401 on 2002/11/04 by ashishs@fl_ashishs_r400_win

added another vte performance test

Change 61372 on 2002/11/04 by omesh@omesh_r400_linux_marlboro_only_devel

Removed all multiple occurences of Wait_Gfx_Idle(), except the last one, as h/w automatically synchronizes between multiple render states.
Keeping the Wait_Gfx_Idle() between each change of render state would keep the h/w pipeline from being completely full, thus not stressing it
completely.

Change 61356 on 2002/11/04 by ashishs@fl_ashishs_r400_win

another VTE performance test

Change 61335 on 2002/11/04 by omesh@omesh_r400_linux_marlboro_only_devel

Removed some commented code and some dead/junk code. Tested the tiled
test for color_8_8_8_8_number_urepeat and it seemed to work, so I'm
marking the bug as resolved.

Change 61313 on 2002/11/04 by mkelly@fl_mkelly_r400_win_laptop

Commented out texture data write to framebuffer for tiled mode

Change 61234 on 2002/11/02 by smoss@smoss_crayola_linux_orl_emu_regress

new path for output

Change 61198 on 2002/11/01 by omesh@omesh_r400_linux_marlboro_only_devel

Completed the Various Format testcases in which each color buffer is
programmed to have a different (or random) color format. Also introduced
6 more random testcases. Visually verified the results, except atleast
COLOR_8_A which doesn't render correctly in any buffer. There's already
a bug filed for this symptom (Bugzilla Bug I.D. 407)

Change 61166 on 2002/11/01 by ashishs@fl_ashishs_r400_win

updated

Change 61161 on 2002/11/01 by ashishs@fl_ashishs_r400_win

updated

Change 61156 on 2002/11/01 by ashishs@fl_ashishs_r400_win

to test vte control register and also varying display dimesions for texture mapped primitive.

Change 61142 on 2002/11/01 by ashishs@fl_ashishs_r400_win

VTE performance test

Change 61132 on 2002/11/01 by csampayo@fl_csampayo2_r400

Removed events from within partial packets

Change 61106 on 2002/11/01 by omesh@omesh_r400_linux_marlboro_only_devel

Added overflow/underflow versions of the RB format tests to test RB
clamping. As of now, the emulator seems to have a clamping bug. It does
not seem to be doing any clamping, so I will file a Bugzilla on it.
Here's another 89 testcases.

Change 61099 on 2002/11/01 by mkelly@fl_mkelly_r400_win_laptop

Clamp bottom right XY scissor to DISP_X_DIM and DISP_Y_DIM

Change 61079 on 2002/11/01 by vgoel@fl_vgoel2

added r400vgt_hos_PNT_adaptive

Change 61076 on 2002/11/01 by vgoel@fl_vgoel2

added tone ampping and RTL tests

Change 61069 on 2002/11/01 by vgoel@fl_vgoel2

RTL extended to render 228 objects

Change 61067 on 2002/11/01 by vgoel@fl_vgoel2

reduced the image size to 128 x 128

Change 61043 on 2002/11/01 by georgev@devel_georgev_r400_lin2_marlboro

Update tests so they match changes to z buffer.

Change 61027 on 2002/11/01 by llefebvr@llefebvre_laptop_r400_emu

Flipped GPR_MANAGEMENT fields to match CP.
Removed float to fix on the XY load.
Modified tests that used the feature accordingly.

Change 61018 on 2002/11/01 by ashishs@fl_ashishs_r400_win

adding more frustum triangle clipping tests.

Change 60993 on 2002/11/01 by jhoule@jhoule_r400_win_marlboro

Changed ACrYCbBlack to ACbYCrBlack and ACrCbYBlack to ACbCrYBlack.
In other words, swapped Cr and Cb.
This was changed in version 1.57 of the Instr/Const.

Change 60949 on 2002/11/01 by mkelly@fl_mkelly_r400_win_laptop

Kill param test...

Change 60937 on 2002/11/01 by mkelly@fl_mkelly_r400_win_laptop

Broken out line list cases from parameterized test
Update tracker

Change 60841 on 2002/10/31 by bbuchner@fl_bbuchner_r400_win

updated memory allocations

Change 60826 on 2002/10/31 by vgoel@fl_vgoel2

added new PNT complex adaptive test

Change 60810 on 2002/10/31 by csampayo@fl_csampayo_r400

Added new SU test, updated test_list and test tracker accordingly

Change 60764 on 2002/10/31 by smoss@smoss_crayola_linux_orl_regress

another new path for linux

Change 60758 on 2002/10/31 by mkelly@fl_mkelly_r400_win_laptop

Changing test r400sc_parameterized_line_list_01
from parameterized to single case per test

Change 60740 on 2002/10/31 by bbuchner@fl_bbuchner_r400_win

subdivision facet test and vtx shader

Change 60731 on 2002/10/31 by smoss@smoss_crayola_linux_orl_regress

update unix vector path

Change 60671 on 2002/10/31 by ashishs@fl_ashishs_r400_win

added render_engine.Wait_Gfx_Idle_No_Flush();

Change 60663 on 2002/10/31 by kevino@kevino_r400_linux_marlboro

Updated to accept multiple input values, and a scale value

Change 60654 on 2002/10/31 by mkelly@fl_mkelly_r400_win_laptop

Update test to ensure msaa_num_samples is not set to 4 or 6.
Fix runp dumpdiff paths

Change 60644 on 2002/10/31 by kevino@kevino_r400_linux_marlboro

Converts unsigned in version of float to float and shows value * 255 in hex and decimal

Change 60628 on 2002/10/31 by smoss@smoss_crayola_win

su tests

Change 60622 on 2002/10/31 by kevino@kevino_r400_win_marlboro

Updated to 10/31 results

Change 60600 on 2002/10/31 by mkelly@fl_mkelly_r400_win_laptop

Comment out test r400cl_gband_tcl_01 until we resolve the hang issue

Change 60528 on 2002/10/30 by vgoel@fl_vgoel2

corrected shader to compute correct tessellation factor and added support to export to two different base addresses.

Change 60516 on 2002/10/30 by georgev@devel_georgev_r400_lin2_marlboro

Fix Range to RangeF conversion.

Change 60469 on 2002/10/30 by ashishs@fl_ashishs_r400_win

To test the VTE control register
Cycles unique permutations of the VTE control register
Condensed test which runs (vap_vte_00 - vap_vte_09 of R200) in one test.

Change 60439 on 2002/10/30 by mkelly@fl_mkelly_r400_win_laptop

Remove ALL binary and dump files from previous run before next run.

Change 60438 on 2002/10/30 by mkelly@fl_mkelly_r400_win_laptop

New test for SC window scissor...

Change 60347 on 2002/10/30 by georgev@devel_georgev_r400_lin2_marlboro

Fix unrepeatable random bug.

Change 60270 on 2002/10/30 by frivas@FL_FRivas

Update to test to correctly determine DMA size.

Change 60260 on 2002/10/30 by kevino@kevino_r400_win_marlboro

Added tex size setting tp tp_all testcase files. Added tp_unsigned32_01_stmap that uses tex coords in map.

Change 60253 on 2002/10/30 by mkelly@fl_mkelly_r400_win_laptop

Update compare_list with new dumps
Modify clean_previous_test_run() to work with parameterized tests

Change 60234 on 2002/10/30 by smoss@smoss_crayola_linux_orl_emu_regress

gold path for unix

Change 60124 on 2002/10/29 by vgoel@fl_vgoel2

changed rendering of image to floating point pixel rendering and modified scene data to float type.

Change 60110 on 2002/10/29 by vgoel@fl_vgoel2

updated to to rect patches tessellation of 11.5

Change 60086 on 2002/10/29 by ashishs@fl_ashishs_r400_win

updated

Change 60076 on 2002/10/29 by mkelly@fl_mkelly_r400_win_laptop

* runp: clean intermediate files, add -v (dumpdiff) option
* finish multi-prim scissor para test
* update test_list_parameterized for new test cases

Change 60074 on 2002/10/29 by ashishs@fl_ashishs_r400_win

To test the VTE control register
Cycles all permutations of the VTE control register
Currently this test has slight(very minor) matching problem with the
corresponding R200 test when 1 texture is enabled.(under review)

Change 60062 on 2002/10/29 by kevino@kevino_r400_win_marlboro

added tfc print

Change 60058 on 2002/10/29 by kevino@kevino_r400_win_marlboro

Got rid of randomize base seed- Paul Mitchel said it shouldn't be in tests.

Change 60020 on 2002/10/29 by omesh@omesh_r400_linux_marlboro_only_devel

Fixed a typo.

Change 60014 on 2002/10/29 by frivas@FL_FRivas

Fixed a few bugs with test. It now alternates fully between RPatch, TPatch, and LPatch rendering.

Change 60008 on 2002/10/29 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 59928 on 2002/10/29 by omesh@omesh_r400_linux_marlboro_only_devel

Removed all occurences of randomizing the Base Seed within a test, as
this can be done at the Makefile level or runtime, when the test is run.
This allows for the Base Seed to be logged and the results can be
reproduced.

Change 59927 on 2002/10/29 by vgoel@fl_vgoel2

modified for multiple objects and multiple spot lights

Change 59920 on 2002/10/29 by omesh@omesh_r400_linux_marlboro_only_devel

Added PA_SC_AA_CONFIG_max_sample_dist programming of 0x8 needed for
debug of the sample mask bug for the basic multisample testcase.

Change 59912 on 2002/10/29 by kevino@kevino_r400_win_marlboro

Dump texture to ppm and at end

Change 59855 on 2002/10/28 by vgoel@fl_vgoel2

updated to display 28 objects

Change 59810 on 2002/10/28 by frivas@FL_FRivas

Initial check-in of HOS RTL test. Test switches between Rect, Tri, and Line patches. It implements texture mapping, lighting, and varies tessellation and reuse 1-14.5 and 4-6, respectively. This test is not yet ready to be used in any regression.

Change 59787 on 2002/10/28 by omesh@omesh_r400_linux_marlboro_only_devel

Added the basic multisample test that Bill Lawless is working with. This is so that the
people handling the Quad Mask bug can try this test against which the Bugzilla report is
probably filed, by Bill.

Change 59746 on 2002/10/28 by hwise@fl_hwise_r400_win

Add include paths to primlib files so the test would compile

Change 59724 on 2002/10/28 by mkelly@fl_mkelly_r400_win_laptop

Added -v option for using dumpdiff instead of byte compare for framebuf

Change 59686 on 2002/10/28 by smoss@regress_crayola_linux_orl_emu_regress

test

Change 59685 on 2002/10/28 by omesh@omesh_r400_linux_marlboro_only_devel

Added ROP3 tests (29) to rbrc and gc testbenches.
Re-enabled Z tests in the rbrc testbench.
Ran some ROP3 tests, although not all and the results looked good on the emulator.
Changed format of tests to 8888 to get greater coverage.

Change 59667 on 2002/10/28 by smoss@regress_crayola_linux_orl_emu_regress

test

Change 59663 on 2002/10/28 by kevino@kevino_r400_win_marlboro

Same test as tp_simple_02, but uses buildLevel to generate map that has coords in map

Change 59660 on 2002/10/28 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 59592 on 2002/10/27 by csampayo@fl_csampayo_lt_r400

Updated test_list and test tracker status for the following tests:
r400vgt_multi_pass_pix_shader_01
r400vgt_multi_pass_pix_shader_02
r400vgt_multi_pass_pix_shader_03
r400vgt_multi_pass_pix_shader_04
r400vgt_multi_pass_pix_shader_05
r400vgt_multi_pass_pix_shader_06

Change 59564 on 2002/10/27 by smoss@smoss_crayola_linux_orl_regress

temp file

Change 59478 on 2002/10/25 by csampayo@fl_csampayo2_r400

Add new multi-pass pixel shader test

Change 59476 on 2002/10/25 by csampayo@fl_csampayo2_r400

Increased framebuffer size and general cleanup

Change 59437 on 2002/10/25 by ashishs@fl_ashishs_r400_win

update

Change 59428 on 2002/10/25 by vromaker@vromaker_r400_linux_marlboro

updates

Change 59426 on 2002/10/25 by ashishs@fl_ashishs_r400_win

Adding more guard band tests. primitives wih triangle strip, quad list, and quad strip. Each test has 4 quadrants with each quadrant having its primitive clipped or discarded according to guardband settings. 1st quadrant the primitive is gourarad shaded, 2nd and 3rd quadrant the primitive is flat shaded with start and end vertex as provoking vertex respectively and the 4th quadrant has the primitive mapped with 6 texture maps.

Change 59381 on 2002/10/25 by kevino@kevino_r400_win_marlboro

updated 1D test reason and tp_unsigned32_01

Change 59355 on 2002/10/25 by kevino@kevino_r400_win_marlboro

Updated 1D failure reason and tp_unsigned32_01 hang info

Change 59344 on 2002/10/25 by tien@ma_spinach

Updated for clamp cases tp_simple_01.

Change 59318 on 2002/10/25 by frivas@FL_FRivas

Update to Matrix_Class definition to allow test to compile under Linux.

Change 59313 on 2002/10/25 by csampayo@fl_csampayo2_r400

* Updated the framebuffer size to accomodate allocated buffers
* Changed texture base address
* Revised multi-pass control event placement
* Revised PV calculation

Change 59310 on 2002/10/25 by kevino@kevino_r400_win_marlboro

update 1D tp_simple_01 to emu_fail

Change 59303 on 2002/10/25 by csampayo@fl_csampayo2_r400

Adjusted framebuffer size to accomodate allocated buffers. Moved texture base address

Change 59300 on 2002/10/25 by ashishs@fl_ashishs_r400_win

more gband tests...

Change 59228 on 2002/10/24 by tien@ma_spinach

Updated tp_simple_01_stmap status

Change 59211 on 2002/10/24 by ashishs@fl_ashishs_r400_win

update

Change 59207 on 2002/10/24 by ashishs@fl_ashishs_r400_win

Gband tests. 4 primitives, each in one of the 4 quadrants. primitive 1 with gourard shading, primitive 2 with flat shading with start vertex as provoking vtx and primitve 3 with flat

shading and LAST vtx as the provoking vtx and 4th primitive with 6 textures. Each test has a different setting of the Guard bands and also differ in type viz Triangle STRIP or Triangle FAN

Change 59157 on 2002/10/24 by mkelly@fl_mkelly_r400_win_laptop

Update with new SC para tests...

Change 59100 on 2002/10/24 by ashishs@fl_ashishs_r400_win

update

Change 59095 on 2002/10/24 by ashishs@fl_ashishs_r400_win

Test Purpose:Tests guard band clipping (LINE STRIP)

Change 59094 on 2002/10/24 by ashishs@fl_ashishs_r400_win

Test Purpose:Tests guard band clipping (LINE LIST)

Change 59093 on 2002/10/24 by ashishs@fl_ashishs_r400_win

Test Purpose: Tests guard band clipping (POINT LIST)

Change 59090 on 2002/10/24 by ashishs@fl_ashishs_r400_win

Description:Tests guard band clipping (TRIANGLE_LIST)

Method:places vertices at various combinations of positions
relative to the discard guard band and the clip guard band. The positions are assigned variable names as follows:
Position 0: outside the window, but w/in the discard guard band
Position 1: outside the discard band, but w/in the clipping guard band
Position 2: outside the clipping guard band

Expected Results:The test should have polygons with the vertices in the following
positions:

```
 Vertex        0 1 2      Result
 -----------------------------------------------------
 Position      0 0 0      No clipping
 0 0 1      No clipping
 0 0 2      Vertex 2 clipped to guard band
 0 1 0      No clipping
 0 1 1      No clipping
 0 1 2      Vertex 2 clipped to guard band
 0 2 0      Vertex 1 clipped to guard band
 0 2 1      Vertex 1 clipped to guard band
 0 2 2      Vertex 1 & 2 clipped to guard band
 0 0 0      No clipping
 1 0 1      No clipping
```

```
 1 0 2      Vertex 2 clipped to guard band
 1 1 0      No clipping
 1 1 1      Discard polygon
 1 1 2      Discard polygon
 1 2 0      Vertex 1 clipped to guard band
 1 2 1      Discard polygon
 1 2 2      Discard polygon
 2 0 0      Vertex 0 clipped to guard band
 2 0 1      Vertex 0 clipped to guard band
 2 0 2      Vertex 0 & 2 clipped to guard band
 2 1 0      Vertex 0 clipped to guard band
 2 1 1      Discard polygon
 2 1 2      Discard polygon
 2 2 0      Vertex 0 & 1 clipped to guard band
 2 2 1      Discard polygon
 2 2 2      Discard polygon
```

Change 59077 on 2002/10/24 by llefebvr@llefebvre_laptop_r400_emu

Fixing a problem in the SQ RS management where the call return address wasn't save correctly in some cases. Fixes the advanced_test.

Change 59053 on 2002/10/24 by mkelly@fl_mkelly_r400_win_laptop

SC Clip Rect tests, LINE_LOOP, STIPPLE, FSAA permutations...

Change 58992 on 2002/10/23 by csampayo@fl_csampayo_r400

Adding multi-pass pixel shader with SC processing enabled

Change 58990 on 2002/10/23 by tien@ma_spinach

Just added something I was looking at

Change 58987 on 2002/10/23 by csampayo@fl_csampayo_r400

Adding another multi-pass pixelshader test

Change 58981 on 2002/10/23 by vgoel@fl_vgoel2

changed the vertex format register for vec_1 in pass 2

Change 58970 on 2002/10/23 by ashishs@fl_ashishs_r400_win

Tests guard band clipping
4 primitives in 4 quadrants.
Each primitive has 6 vertices and first vert on vertical guard band.

Top Left Quadrant, gouraud shading

Top Right Quadrant, FLAT SHADING with START vertex as provoking vtx
Lower Right Quadrant, FLAT SHADING with END vertex as provoking vtx
Lower Left Quadrant, SIX TEXTURES, with COLOR0 as transparent

Method: 6 Vert Strip, first vert on vertical guard band
Expected Results: 4 primitives, four quadrants, no clipping, no discarding

Change 58927 on 2002/10/23 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 58893 on 2002/10/23 by abeaudin@abeaudin_r400_win_marlboro

fixed program so that it does not abort

Change 58815 on 2002/10/23 by mkelly@fl_mkelly_r400_win_laptop

All 16 permutations of CLIP RECTS tested...

Change 58813 on 2002/10/23 by vgoel@fl_vgoel2

made changes to incorprate aperture for memory export and adding offsets

Change 58735 on 2002/10/22 by vgoel@fl_vgoel2

added texture map with spot light

Change 58719 on 2002/10/22 by vgoel@fl_vgoel2

GI changes , includes texture mapping

Change 58705 on 2002/10/22 by subad@subad_r400_win_marlboro

looking at standard testcase

Change 58697 on 2002/10/22 by smoss@smoss_crayola_win

su tests

Change 58693 on 2002/10/22 by vgoel@fl_vgoel2

modified scene data

Change 58678 on 2002/10/22 by csampayo@fl_csampayo_r400

Some clean up and minor updates

Change 58636 on 2002/10/22 by vgoel@fl_vgoel2

changed vtx data type to float from double

Change 58634 on 2002/10/22 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint on clip rect permutations...

Change 58614 on 2002/10/22 by ygiang@ygiang_r400_win_marlboro_p4

added: new opcode sp tests and testcases for existing tests

Change 58609 on 2002/10/22 by kevino@kevino_r400_win_marlboro

Added fmt2101010 to tp_multitexture_01

Change 58571 on 2002/10/22 by vgoel@fl_vgoel2

added test scene for GI

Change 58557 on 2002/10/22 by llefebvr@llefebvre_laptop_r400_emu

resetting the wrong count.

Change 58548 on 2002/10/22 by csampayo@fl_csampayo2_r400

Adding another multi-pass pixel shader test

Change 58541 on 2002/10/22 by subad@subad_r400_win_marlboro

updated

Change 58516 on 2002/10/22 by mkelly@fl_mkelly_r400_win_laptop

Update scripts for LSF

Change 58431 on 2002/10/21 by csampayo@fl_csampayo2_r400

Update per last memory export scheme

Change 58423 on 2002/10/21 by csampayo@fl_csampayo2_r400

Update per latest memory export scheme

Change 58396 on 2002/10/21 by kevino@kevino_r400_win_marlboro

Added tm.prepare(tfc) to test

Change 58381 on 2002/10/21 by llefebvr@llefebvre_laptop_r400_emu

Memory export architectural change to get rid of the normalize in the SX (and do it in the SP).

Change 58355 on 2002/10/21 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint on load sharing test emulation scripts...

Change 58333 on 2002/10/21 by llefebvr@llefebvre_laptop_r400_emu

Fixing multi-pass shader test.

Change 58315 on 2002/10/21 by ygiang@ygiang_r400_win_marlboro_p4

fixed: random test case for GPRS

Change 58314 on 2002/10/21 by georgev@devel_georgevhw_r400_lin_marlboro

Fixed out of range loop increment (126 instead of 127

Change 58310 on 2002/10/21 by kryan@kryan_r400_win_marlboro

test_lib/src/chip/gfx/.../Makefile

test_lib/src/chip/sys/cp/Makefile

test_lib/src/chip/perf/Makefile

test_lib/scripts/results_diff.pl

- Added REGRESS_DIFF_OPTIONS variable in results_diff.pl that controls whether or not the dumpfile headers are compared. It is set in the Makefile on a per directory basis for the Marlborough and Orlando test directories.

export REGRESS_DIFF_OPTIONS = IGNORE_DUMP_HEADERS

This will not affect Toronto tests since they do not use PrimLib for creating dumpfiles, and this variable was not set in the Toronto test suite directories.

fill_dump.cpp

- Fixed get_FB_start(),get_FB_size(), get_AGP_start(), get_AGP_size() functions to correctly read registers and return the values. Previously they were using the register spec from previous chips which required additional operations to determine the correct value.

DEPTH_SURFACE

- Begin work on DEPTH_SURFACE class to handle fill/dump of Depth Surfaces. Work in progress.

/cmn_lib/tools/autoreg/gen_testdll/gen_ar_test_render_registers.cpp

RENDER_REGISTERS

- Added copy constructor and assignment constructor (operator=) for this class. This will copy all member data (ie. register values from source object to the current object.

RENDER_REGISTER_STATE

- Added copy constructor and assignment constructor (operator=) for this class. This will copy all member data, allocating new copies for dynamically allocated memory when necessary. This will update the register_map to include any registers that have been modified for this object.

The incremental register stream is not copied since this is built when the Get_Incremental_Register_Stream() function is called.

- Added Set_Destination_Base(uint32 memory_area_offset, uint32 color_surface_index, MEMORY_AREA& memory_area) function that will default to ColorSurface0, and the FrameBuffer as the memory area. It will add the offset

to the base address of the memory aperture to determine the final full device

address for the RB_COLORn_BASE register. It will also make sure that the

base address is aligned properly.

RENDER_STATE

- Work on copy constructor and assignment constructor (operator=). Since

it is not complete, throw error if someone uses it.

Change 58307 on 2002/10/21 by csampayo@fl_csampayo_lt_r400

Cleaned up tests, added 1 new VGT test to test_list and updated the test tracker accordingly

Change 58295 on 2002/10/21 by csampayo@fl_csampayo_lt_r400

Updated test description

Change 58294 on 2002/10/21 by kevino@kevino_r400_win_marlboro

Added texture dump function

Change 58292 on 2002/10/21 by csampayo@fl_csampayo_r400

Updated comments

Change 58278 on 2002/10/21 by csampayo@fl_csampayo2_r400

Adding new multi pass pixel shader test

Change 58275 on 2002/10/21 by csampayo@fl_csampayo2_r400

Update test description and clean up shaders

Change 58245 on 2002/10/21 by kevino@kevino_r400_win_marlboro

updated based on 10/21 regressions

Change 58105 on 2002/10/18 by kevino@kevino_r400_win_marlboro

Updated test case MipMinMag_BaseMapLinearLinear_4x4_LL

Change 58103 on 2002/10/18 by tien@ma_spinach

Added a description for one of the tests.

Change 58100 on 2002/10/18 by csampayo@fl_csampayo2_r400

Update for export memory aperture. Expand on shader comments

Change 58099 on 2002/10/18 by omesh@omesh_r400_linux_marlboro_only_devel

Added the modified ROP3 tests (29 testcases) to Perforce. However, the emulator now seems to
assert for the FMT_4_4_4_4 format (Which I think it shouldn't). I am also filing a Bugzilla
report on this. Also added a modified Makefile. Some of the tests included in the Makefile only exist in
my own workspace, but this should not hurt any builds/runs of either the emulator or the other
tests.

Change 58092 on 2002/10/18 by csampayo@fl_csampayo2_r400

Update some comments

Change 58089 on 2002/10/18 by csampayo@fl_csampayo2_r400

Updated second export buffer index offset

Change 58084 on 2002/10/18 by csampayo@fl_csampayo2_r400

Fixed typo

Change 58071 on 2002/10/18 by csampayo@fl_csampayo_r400

Updated for export memory aperture

Change 58063 on 2002/10/18 by kevino@kevino_r400_win_marlboro

Initial checkin

Change 58049 on 2002/10/18 by jhoule@jhoule_r400_win_marlboro

Added full mip packing support.

mip_packing:
Added C library to be shared by driver people.

enum_conv:
Added support for MIP_PACKING field.

tconst:
Added MIP_PACKING field.

tutils:
Changed getMipPitch behavior to honor original 0.
This solves 1D mipmap offset calculation problem in the TC.

texel_position:
Changed texelPosition when border is set to stricty stay within texture dimension.
Border size support still works since it is solely based on the wrap count.

tp:
Added support for mip offset, when appropriate.

texture_manager:
Added mip packing calls.
Added prepare(tfc) function which must be called once and only once prior writing to memory.
Added address printing calls, #ifdef'd with PRINT_ADDRESS.

tp_uber_test:
Supports MIP_PACKING, TRI_JUICE.
Can specify frame buffer size.

Also updated tp regression tests to call prepare(tfc) prior to writing in memory.

Change 58046 on 2002/10/18 by mkelly@fl_mkelly_r400_win_laptop

Perl script which runs one r400 test and behaves like the script "regress_r400"
This will be used for full regressions with LSF
A master script will call this script for each test in "test_list" and tally statistics

Change 58034 on 2002/10/18 by llefebvr@llefebvre_laptop_r400_emu

Added aperture checks.

Change 58005 on 2002/10/18 by llefebvr@llefebvre_laptop_r400_emu

Fixing r400vgt_vtx_export_very_very_simple_04 memory export test.

Change 57980 on 2002/10/18 by vgoel@fl_vgoel2

modifed files for recent vertex export changes

Change 57971 on 2002/10/18 by vgoel@fl_vgoel2

New vertex export test exporting to 2 different buffers

Change 57962 on 2002/10/18 by csampayo@fl_csampayo_r400

New vtx export case

Change 57939 on 2002/10/18 by ashishs@fl_ashishs_r400_win

updated test_list and Tracker for the corresponding change in file name viz r400cl_gabnd_04 to r400cl_gband_06

Change 57935 on 2002/10/18 by ashishs@fl_ashishs_r400_win

Deleted test r400cl_gband_04 and renamed it to r400cl_gband_06. Also updated the Validation_Approach_plan.doc since it had some errors with tests name and numberings.

Change 57896 on 2002/10/18 by mkelly@fl_mkelly_r400_win_laptop

Update version #

Change 57816 on 2002/10/17 by csampayo@fl_csampayo_r400

Update proves that SQ generated indices work but, pixel exports may not be working correctly

Change 57803 on 2002/10/17 by vgoel@fl_vgoel2

added shader for spot light for GI

Change 57801 on 2002/10/17 by ygiang@ygiang_r400_win_marlboro_p4

added: ; to sp command

Change 57800 on 2002/10/17 by ygiang@ygiang_r400_win_marlboro_p4

resubmit

Change 57797 on 2002/10/17 by ygiang@ygiang_r400_win_marlboro_p4

more sp boundary and fixed for frame buffer image

Change 57790 on 2002/10/17 by ygiang@ygiang_r400_win_marlboro_p4

added: more sp boundary tests

Change 57780 on 2002/10/17 by csampayo@fl_csampayo_r400

Simplified to just 1 case

Change 57760 on 2002/10/17 by kevino@kevino_r400_win_marlboro

Added 3D map clamp tests

Change 57754 on 2002/10/17 by mkelly@fl_mkelly_r400_win_laptop

Add ability to define files to compare against gold on a per-test
basis. See test_list file in test_lib/src/chip/sys/cp for an example.

Change 57752 on 2002/10/17 by ygiang@ygiang_r400_win_marlboro_p4

fixed:alu const var

Change 57731 on 2002/10/17 by vgoel@fl_vgoel2

added GI test files

Change 57705 on 2002/10/17 by ygiang@ygiang_r400_win_marlboro_p4

fixed:comments

Change 57703 on 2002/10/17 by ygiang@ygiang_r400_win_marlboro_p4

added: boundary abs opcode test

Change 57692 on 2002/10/17 by omesh@omesh_r400_linux_marlboro_only_devel

Added an additional triangle to testcase to demonstrate fog not working
to Alicia, for fog factor = 1.0. I still get a fully fogged triangle.

Change 57678 on 2002/10/17 by kevino@kevino_r400_win_marlboro

Same as tp_simple_01, but uses ST coords in map w/ tm's buildLevel_STmap_8888 function

Change 57652 on 2002/10/17 by csampayo@fl_csampayo_r400

Fix texture base address and add missing vertex shader

Change 57602 on 2002/10/16 by csampayo@fl_csampayo_r400

Initial check in for multi-pass pixel shader test

Change 57522 on 2002/10/16 by smoss@smoss_crayola_linux_orl_regress

all tests so far

Change 57510 on 2002/10/16 by csampayo@fl_csampayo_r400

Added 1 new HOS with index reset test, updated test_list and test tracker

Change 57478 on 2002/10/16 by mkelly@fl_mkelly_r400_win_laptop

Add finding and statistics handling for CP tests which begin with "e2"

Change 57477 on 2002/10/16 by smoss@smoss_crayola_win

su tests

Change 57468 on 2002/10/16 by ashishs@fl_ashishs_r400_win

updated

Change 57446 on 2002/10/16 by mkelly@fl_mkelly_r400_win_laptop

Parameterized LINE_LIST, random combinations of PA settings...

Change 57445 on 2002/10/16 by ashishs@fl_ashishs_r400_win

adding more CL UCP combos tests

Change 57425 on 2002/10/16 by ashishs@fl_ashishs_r400_win

updated

Change 57415 on 2002/10/16 by ashishs@fl_ashishs_r400_win

test edited to include more combinations

Change 57318 on 2002/10/16 by mkelly@fl_mkelly_r400_win_laptop

Update shader filenames...

Change 57313 on 2002/10/16 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 57310 on 2002/10/16 by mkelly@fl_mkelly_r400_win_laptop

Reduce number of indices in tests from 1024 to 256.

Change 57257 on 2002/10/15 by omesh@omesh_r400_linux_marlboro_release

Changed test to match latest RB register specs eliminating the higher 32 bit mask for an originally 64 bit mask.
Also removed all references to any format > 32 bits per pixel. Removed a total of 12 testcases. Regression nightly run
list has already been updated with these 12 test cases removed.

Change 57247 on 2002/10/15 by csampayo@fl_csampayo_r400

Added 5 new VGT index reset tests and updated test_list and test tracker

Change 57222 on 2002/10/15 by ashishs@fl_ashishs_r400_win

updated

Change 57202 on 2002/10/15 by csampayo@fl_csampayo_r400

Updated test_list and test tracker for the following VGT tests:
r400vgt_vtx_export_very_very_simple_01
r400vgt_vtx_export_very_very_simple_02
r400vgt_vtx_export_very_very_simple_03

Change 57200 on 2002/10/15 by ashishs@fl_ashishs_r400_win

updated

Change 57193 on 2002/10/15 by csampayo@fl_csampayo2_r400

Update description and framebuffer size

Change 57179 on 2002/10/15 by mkelly@fl_mkelly_r400_win_laptop

Locate many of each primtype randomly through all of hardware space in 12 TEST_CASEs
Clipping enabled and Scissor clipping to window.

Change 57149 on 2002/10/15 by llefebvr@llefebvre_laptop_r400_emu

only writing to EM0 now.

Change 57143 on 2002/10/15 by csampayo@fl_csampayo2_r400

Updated vtx export tests and added a new case

Change 57121 on 2002/10/15 by kevino@kevino_r400_win_marlboro

Added simple 3D test case. 3x3x3 volume map is read from texture files volumemap_3x3_n.ppm.
Mapped onto 9x3 rectangle. T goes 0 to 1. S goes 0 to 3 and R goes 0 to 1, so should see each layer of map.

Change 57107 on 2002/10/15 by kevino@kevino_r400_win_marlboro

Updated golden files. Old ones had info in rd_r for some reason.

Change 57093 on 2002/10/15 by ashishs@fl_ashishs_r400_win

update

Change 57076 on 2002/10/15 by llefebvr@llefebvre_laptop_r400_emu

Was rotating the mask in SX and shouldn't (memory exports).

Change 56985 on 2002/10/14 by kevino@kevino_r400_win_marlboro

get rid of tex map dump

Change 56984 on 2002/10/14 by smoss@smoss_crayola_win

multi-context phase III

Change 56977 on 2002/10/14 by ashishs@fl_ashishs_r400_win

update

Change 56974 on 2002/10/14 by ashishs@fl_ashishs_r400_win

Features Tested:frustum clipping; LFT rotated to each of 8 corners
Test Purpose:LFT rotated to each of 8 corners of clipping frustum
- 51*8 polygons
Expected Results:clipped, overlapping triangles forming polygons in each ordinal
direction (E, W, S, N, NW, NE, etc.); several overlapping triangles (forming polygons) in center of image

Change 56957 on 2002/10/14 by kevino@kevino_r400_win_marlboro

Update golden files.

Change 56943 on 2002/10/14 by kevino@kevino_r400_win_marlboro

Added more golden images

Change 56936 on 2002/10/14 by csampayo@fl_csampayo_r400

Added 3 new VGT tests, updated test_list and test tracker

Change 56932 on 2002/10/14 by llefebvr@llefebvre_laptop_r400_emu

Fixing addressing errors in tests and SX block for memory exports.

Change 56930 on 2002/10/14 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 56927 on 2002/10/14 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 56915 on 2002/10/14 by mkelly@fl_mkelly_r400_win_laptop

Expand SC parameterized general random

Change 56892 on 2002/10/14 by mkelly@fl_mkelly_r400_win_laptop

SC parameterized random tests

Change 56889 on 2002/10/14 by csampayo@fl_csampayo_lt_r400

Updated for proper event write mode

Change 56879 on 2002/10/14 by kevino@kevino_r400_win_marlboro

Changes for buildLevel (instead of reaLevel) to generate texture map

Change 56878 on 2002/10/14 by kevino@kevino_r400_win_marlboro

Generated by make tp_simple_01.golden

Change 56876 on 2002/10/14 by kevino@kevino_r400_win_marlboro

Golden images from emu for tp_simple_01 standard case

Change 56855 on 2002/10/14 by smoss@smoss_crayola_win

su tests

Change 56834 on 2002/10/14 by mkelly@fl_mkelly_r400_win_laptop

Comment out a print

Change 56723 on 2002/10/11 by csampayo@fl_csampayo2_r400

Update test for new event writing mode and remove from uncommented list

Change 56707 on 2002/10/11 by csampayo@fl_csampayo_r400

Update for new event writing mode

Change 56670 on 2002/10/11 by ashishs@fl_ashishs_r400_win

CL test:
LFT corner clip focus
Test Purpose:to test LFT corner clips - 51 polygons
Expected Results: 4 sets of clipped gourand-shaded polygons in each corner of the viewport

Change 56658 on 2002/10/11 by georgev@devel_georgevhw_r400_lin_marlboro

Addition of new tests that don't work.

Change 56635 on 2002/10/11 by mkelly@fl_mkelly_r400_win_laptop

Full PA(CL/VTE/SU/SC) control via parameterized template

Change 56619 on 2002/10/11 by kevino@kevino_r400_win_marlboro

uber_map- in pack function, if nH=0, set nH to 1 sso go through loop once. (tm.buildLevel causes this)
tp_simple_01 - set up 1D test cases as well as ones where S or T constant and the other ranges from 0-1 or 0-0.5

Change 56602 on 2002/10/11 by csampayo@fl_csampayo2_r400

Updated to export meaningful floats

Change 56561 on 2002/10/11 by hwise@fl_hwise_r400_win

CP Emulator Updates (for Real-Time Streams)
1) Added register read/write support for these registers:
a) CP_RT_BASE
b) CP_RT_BUFSZ
c) CP_RT_ST_BASE
d) CP_RT_ST_BUFSZ
2) Added logic to Pre-Fetch Parser to do real-time fetches and send fetched data to appropriate Re-Ordering Queue
3) Updated CP Microengine for better debugability of the real-time instance (tile)
4) Fixed bug in CP Microengine for setting the "incremental_update" boolean
5) Added logic to correctly set the "rt_enabled" boolean (was always set to zero)
6) Fixed bug in the Real-Time Event Engine for dis-arming and event at the correct cycle
7) Fixed bug in the Real-Time Event Engine "Compare Function" to return the "poll_valid" input signal when compare is successful (was always returning "true")

PrimLib / Promo4Lib Updates
1) Changed <BLK>.<REG>.read() to <BLK>.<REG>.read_nochk() in memory_area.cpp so these register reads are not considered for comparison
2) Added Real-Time wait functions to plgx namespace
a) void plgx::wait_gfx_rt_idle(void)
b) bool plgx::wait_gfx_rt_idle_timeout(void)
c) void plgx::wait_cp_rt_idle(void)

d) bool plgx::wait_cp_rt_idle_timeout(void)
3) Increased PCI clocking from 0 to 50 clocks at end of the function void plgx::wait_dma_idle(void)
4) Increased MAX_RB_BUF_SIZE to 1048576 dwords (4 MB) which is used to define maximum size for the ring and indirect buffers used by the CP. The default sizes are still 4KB.
5) By default Promo4Lib::init() wil initialize the CP with Real-Time Streams enabled and send ME_INIT packets to both the Non-Real-Time and Real-Time Microengines
5) Added registry key switch to allow user to disable the Real-Time Stream in the Non-Real_time ME_INIT packet (debug use only) "HKEY_LOCAL_MACHINE\\SOFTWARE\\ATI Technologies\ \Debug\\pm4RealTimeInitDisable"
6) Both trace_pm4_packets=="yes" or pm4Verbose!=0 will enable the PM4 stream debug trace to be sent to console (used to only be able to activate this with trace_pm4_packets=="yes")

regress_e Golden Files Updated
* Updated golds since the *.rd_r files no longer contain the register reads introduced by functions in the memory_area.cpp file and the *.mem_framebuf_r header defines changed once real-time streams were enabled by default

Change 56555 on 2002/10/11 by llefebvr@llefebvre_laptop_r400_emu

Fixing the export format in export test.

Change 56547 on 2002/10/11 by mkelly@fl_mkelly_r400_win_laptop

non-parameterized checkpoint...

Change 56487 on 2002/10/10 by csampayo@fl_csampayo2_r400

Adding 32bit float vertex export test

Change 56485 on 2002/10/10 by csampayo@fl_csampayo2_r400

Make this test case into a useful test

Change 56483 on 2002/10/10 by omesh@ma_omesh

Changed back to the older shader to export vertex colors in different orders for all buffers (2 in this case). This is preferable, as apart from only testing the write to different buffers, I also want to check if the exports to each buffer are unique for each channel. The swizzle achieves this.

Change 56455 on 2002/10/10 by jhoule@jhoule_r400_win_marlboro

Same as tp_simple_01.cpp, but using PerfectGradient_4x4.ppm as texture.

Change 56438 on 2002/10/10 by jhoule@jhoule_r400_win_marlboro

More sizes...
R=x position
G=y position

Change 56423 on 2002/10/10 by georgev@devel_georgevhw_r400_lin_marlboro

Added new sq tests for simple conditional debug.

Change 56420 on 2002/10/10 by jhoule@jhoule_r400_win_marlboro

Texture where data correspond to texels, meaning that pixel (10, 13) has 10 in R and 13 in B.
Useful for cache testing.

Change 56415 on 2002/10/10 by mkelly@fl_mkelly_r400_win_laptop

checkpoint...

Change 56409 on 2002/10/10 by vromaker@vromaker_r400_linux_marlboro

update to cond exec

Change 56367 on 2002/10/10 by llefebvr@llefebvre_laptop_r400_emu

Simple memory exports are now working.

Change 56299 on 2002/10/10 by csampayo@fl_csampayo_r400

Added bug test case

Change 56233 on 2002/10/09 by csampayo@fl_csampayo_r400

Updated for multi-context

Change 56171 on 2002/10/09 by csampayo@fl_csampayo_r400

1. Updated tests r400vgt_index_min_max_01 and _02 for multi-context
2. Added new VGT tests r400vgt_index_min_max_03 and _04
3. Updated test_list and the test tracker

Change 56168 on 2002/10/09 by ygiang@ygiang_r400_win_marlboro_p4

added: shader src and dst modifiers tests

Change 56142 on 2002/10/09 by vromaker@vromaker_r400_linux_marlboro

- fix for tp_done/ais_pop collision
- added conditional execution to CFS

Change 56114 on 2002/10/09 by mkelly@fl_mkelly_r400_win_laptop

No outstanding bugs in this directory...

Change 56108 on 2002/10/09 by mkelly@fl_mkelly_r400_win_laptop

Cleaning up old bugs, keeping some as valid tests in related block...

Change 56104 on 2002/10/09 by mkelly@fl_mkelly_r400_win_laptop

Move a test from bug status to good block test for the SC...

Change 56099 on 2002/10/09 by mkelly@fl_mkelly_r400_win_laptop

Move from bugs to permanent test in CL block

Change 56097 on 2002/10/09 by mkelly@fl_mkelly_r400_win_laptop

Move nan kill test to CL block

Change 56078 on 2002/10/09 by mkelly@fl_mkelly_r400_win_laptop

Rasterize 224 triangles, 1 packet. Each triangle should hit 4 horizontal quads.
The test moves the triangle 4 quads at a time in X and
2 quads in increasing Y. For each primitive, 4 quads are hit in one tile and the
fourth quad hits the next consecutive tile.

Change 56071 on 2002/10/09 by mkelly@fl_mkelly_r400_win_laptop

Change Wait_Gfx_Idle() to Wait_Gfx_Idle_No_Flush() inside of iterative loops.
Add Wait_Gfx_Idle() before frame buffer dump.

All new tests should contain the Wait_Gfx_Idle() call outside of the test case
generation loop, preferably just before the framebuffer dump. If for some reason,
you are reusing the index and/or vertex buffers in DMA indexing mode, then,
you need to use Wait_Gfx_Idle_No_Flush() within and, Wait_Gfx_Idle() outside the loop.
Also, as you are updating or generating derivative tests, please, update the root
tests as well.

Change 56042 on 2002/10/09 by ygiang@ygiang_r400_win_marlboro_p4

fixed: missing test case definition

Change 56035 on 2002/10/09 by mkelly@fl_mkelly_r400_win_laptop

Uncommented auto shader generation

Change 56033 on 2002/10/09 by mkelly@fl_mkelly_r400_win_laptop

Packed color example usage for VFD

Change 56024 on 2002/10/09 by smoss@smoss_crayola_linux_orl_regress

modifications for Linux, getting extra path stuff

Change 56018 on 2002/10/09 by mkelly@fl_mkelly_r400_win_laptop

Missed update for idle..

Change 56017 on 2002/10/09 by mkelly@fl_mkelly_r400_win_laptop

Change Wait_Gfx_Idle() to Wait_Gfx_Idle_No_Flush() inside of iterative loops.
Add Wait_Gfx_Idle() before frame buffer dump.

All new tests should contain the Wait_Gfx_Idle() call outside of the test case
generation loop, preferably just before the framebuffer dump. If for some reason,
you are reusing the index and/or vertex buffers in DMA indexing mode, then,
you need to use Wait_Gfx_Idle_No_Flush() within and, Wait_Gfx_Idle() outside the loop.
Also, as you are updating or generating derivative tests, please, update the root
tests as well.

Change 56008 on 2002/10/09 by mkelly@fl_mkelly_r400_win_laptop

* Rasterize 256 triangles, 1 packet. Each triangle should hit 4 quads.
The test moves the triangle 2 quads at a time in X and
2 quads in increasing Y.
* Update test_list
* Update test documentation in test tracker

Change 56004 on 2002/10/09 by mkelly@fl_mkelly_r400_win_laptop

When -z (zip) option is selected, zip all files except *.mem_framebuf_r

Change 55961 on 2002/10/08 by kevino@kevino_r400_win_marlboro

small prims/textures

Change 55953 on 2002/10/08 by ygiang@ygiang_r400_win_marlboro_p4

added: boundray values test for sp
fixed: image for standard test

Change 55945 on 2002/10/08 by ashishs@fl_ashishs_r400_win

update

Change 55941 on 2002/10/08 by ashishs@fl_ashishs_r400_win

Adding Frustum Line and triangle tests

Change 55938 on 2002/10/08 by kevino@kevino_r400_win_marlboro

Ooops-- the original was actually a 64x32. This is a 96x32

Change 55914 on 2002/10/08 by georgev@devel_georgevhw_r400_lin_marlboro

Some fixes and some new.

Change 55897 on 2002/10/08 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint...

Change 55892 on 2002/10/08 by mkelly@fl_mkelly_r400_win_laptop

Change Wait_Gfx_Idle() to Wait_Gfx_Idle_No_Flush() inside of iterative loops.
Add Wait_Gfx_Idle() before frame buffer dump.

All new tests should contain the Wait_Gfx_Idle() call outside of the test case
generation loop, preferably just before the framebuffer dump. If for some reason,
you are reusing the index and/or vertex buffers in DMA indexing mode, then,
you need to use Wait_Gfx_Idle_No_Flush() within and, Wait_Gfx_Idle() outside the loop.
Also, as you are updating or generating derivative tests, please, update the root
tests as well.

Change 55872 on 2002/10/08 by smoss@smoss_crayola_win

incorrect test names

Change 55869 on 2002/10/08 by smoss@smoss_crayola_win

su tests

Change 55863 on 2002/10/08 by mkelly@fl_mkelly_r400_win_laptop

Change Wait_Gfx_Idle() to Wait_Gfx_Idle_No_Flush() inside of iterative loops.
Add Wait_Gfx_Idle() before frame buffer dump.

All new tests should contain the Wait_Gfx_Idle() call outside of the test case
generation loop, preferably just before the framebuffer dump. If for some reason,
you are reusing the index and/or vertex buffers in DMA indexing mode, then,

you need to use Wait_Gfx_Idle_No_Flush() within and, Wait_Gfx_Idle() outside the loop.
Also, as you are updating or generating derivative tests, please, update the root
tests as well.

Change 55847 on 2002/10/08 by omesh@omesh_r400_linux_marlboro_only_devel

Although Alicia checked in a "no_clamp" version of the pixel shader, it may have been accidentally
the same old versions of the files checked in. I actually removed the clamp from the pixel shader
and also modified the tests. However, although "uinteger" seems fixed, "sinteger" still displays the
bug. Also "srepeat" looks incorrect.

Change 55844 on 2002/10/08 by mkelly@fl_mkelly_r400_win_laptop

Change Wait_Gfx_Idle() to Wait_Gfx_Idle_No_Flush() inside of iterative loops.
Add Wait_Gfx_Idle() before frame buffer dump.

All new tests should contain the Wait_Gfx_Idle() call outside of the test case
generation loop, preferably just before the framebuffer dump. If for some reason,
you are reusing the index and/or vertex buffers in DMA indexing mode, then,
you need to use Wait_Gfx_Idle_No_Flush() within and, Wait_Gfx_Idle() outside the loop.
Also, as you are updating or generating derivative tests, please, update the root
tests as well.

Change 55797 on 2002/10/08 by mkelly@fl_mkelly_r400_win_laptop

Change Wait_Gfx_Idle() to Wait_Gfx_Idle_No_Flush() inside of iterative loops.
Add Wait_Gfx_Idle() before frame buffer dump.

All new tests should contain the Wait_Gfx_Idle() call outside of the test case
generation loop, preferably just before the framebuffer dump. If for some reason,
you are reusing the index and/or vertex buffers in DMA indexing mode, then,
you need to use Wait_Gfx_Idle_No_Flush() within and, Wait_Gfx_Idle() outside the loop.
Also, as you are updating or generating derivative tests, please, update the root
tests as well. The goal is to have as many tests modified as we can, as soon as possible.

Change 55795 on 2002/10/08 by smoss@smoss_crayola_win

su tests

Change 55794 on 2002/10/08 by llefebvr@llefebvre_laptop_r400_emu

Added range check.
Also the test was clamping the values to infinities where clamped prior to the interface.

Change 55790 on 2002/10/08 by kevino@kevino_r400_win_marlboro

Texture created by buildlevel

Change 55775 on 2002/10/08 by mkelly@fl_mkelly_r400_win_laptop

Change Wait_Gfx_Idle() to Wait_Gfx_Idle_No_Flush() inside of iterative loops.
Add Wait_Gfx_Idle() before frame buffer dump.

All new tests should contain the Wait_Gfx_Idle() call outside of the test case
generation loop, preferably just before the framebuffer dump. If for some reason,
you are reusing the index and/or vertex buffers in DMA indexing mode, then,
you need to use Wait_Gfx_Idle_No_Flush() within and, Wait_Gfx_Idle() outside the
loop.
Also, as you are updating or generating derivative tests, please, update the root
tests as well. The goal is to have as many tests modified as we can, as soon as possible.

Change 55730 on 2002/10/07 by csampayo@fl_csampayo_r400

Added high memory addressing bug

Change 55727 on 2002/10/07 by csampayo@fl_csampayo_r400

Added VGT index size test and updated test_list and test tracker

Change 55676 on 2002/10/07 by smoss@smoss_crayola_win

update for multi-context phase 2

Change 55654 on 2002/10/07 by smoss@smoss_crayola_win

regress_e passes on windows

Change 55613 on 2002/10/07 by smoss@smoss_crayola_linux_orl_regress

or not

Change 55612 on 2002/10/07 by smoss@smoss_crayola_linux_orl_regress

taking su offline while I try to get Linux to pass

Change 55604 on 2002/10/07 by mkelly@fl_mkelly_r400_win_laptop

Move para tests to test_list_parameterized

Change 55592 on 2002/10/07 by ygiang@ygiang_r400_win_marlboro_p4

fixed: const alpha value in test

Change 55576 on 2002/10/07 by mkelly@fl_mkelly_r400_win_laptop

Automatically find path where test is located, properly calc. statistics

Change 55575 on 2002/10/07 by abeaudin@abeaudin_r400_win_marlboro

new shader

Change 55567 on 2002/10/07 by smoss@smoss_crayola_win

try this again

Change 55556 on 2002/10/07 by smoss@smoss_crayola_win

try again

Change 55549 on 2002/10/07 by smoss@smoss_crayola_linux_orl_regress

update golds for regress_e

Change 55548 on 2002/10/07 by smoss@smoss_crayola_win

update for multi-context

Change 55535 on 2002/10/07 by mdoggett@mdoggett_r400_linux_local

Added update for corrected RG swap in RB.

Change 55529 on 2002/10/07 by llefebvr@llefebvre_laptop_r400_emu

The mask sent to the RB for memory exports was always 0. This sends the correct mask down.

Change 55528 on 2002/10/07 by jhoule@jhoule_r400_win_marlboro

1D gradient

Change 55527 on 2002/10/07 by smoss@smoss_crayola_win

SU tests

Change 55514 on 2002/10/06 by ashishs@fl_ashishs_r400_win

reverting back these files for an accidental change made to them

Change 55513 on 2002/10/06 by ashishs@fl_ashishs_r400_win

update

Change 55512 on 2002/10/06 by ashishs@fl_ashishs_r400_win

update

Change 55511 on 2002/10/06 by ashishs@fl_ashishs_r400_win

Adding new frustum line tests

Change 55510 on 2002/10/06 by ashishs@fl_ashishs_r400_win

updated for multicontext

Change 55414 on 2002/10/04 by csampayo@fl_csampayo2_r400

Comment out tests that do not run correctly due to bugs

Change 55404 on 2002/10/04 by ygiang@ygiang_r400_win_marlboro_p4

added: vector write mask tests

Change 55376 on 2002/10/04 by ashishs@fl_ashishs_r400_win

updated for multi context

Change 55371 on 2002/10/04 by kevino@kevino_r400_win_marlboro

3D filter tests (need a lot of work)

Change 55365 on 2002/10/04 by kevino@kevino_r400_win_marlboro

Added 3d level testcases, but commented out read3DLevel function call since it is not checked into primlib's TM yet

Change 55355 on 2002/10/04 by mkelly@fl_mkelly_r400_win_laptop

Update multi-chip tests to run test07 as standard, enable SC in para regression script

Change 55350 on 2002/10/04 by kevino@kevino_r400_win_marlboro

Added 1D filter _UUUU testcase so can seperate 1D problems from signed mode problems.

Change 55346 on 2002/10/04 by kevino@kevino_r400_win_marlboro

multiply S and T by 1/Q

Change 55345 on 2002/10/04 by jhoule@jhoule_r400_win_marlboro

Sizes were wrong (8x8 instead of 4x4, 2x2, 1x1).
Created them all for each color.

Change 55338 on 2002/10/04 by kevino@kevino_r400_win_marlboro

test functions

Change 55333 on 2002/10/04 by mkelly@fl_mkelly_r400_win_laptop

Notes

Change 55323 on 2002/10/04 by kevino@kevino_r400_win_marlboro

Changed default mip filter to BaseMap

Change 55312 on 2002/10/04 by ashishs@fl_ashishs_r400_win

test_list updated

Change 55304 on 2002/10/04 by mkelly@fl_mkelly_r400_win_laptop

Add new dump file to regression VgtCcg.dmp for test bench support

Change 55282 on 2002/10/04 by mkelly@fl_mkelly_r400_win_laptop

SC performance test #1

Change 55281 on 2002/10/04 by ygiang@ygiang_r400_win_marlboro_p4

added: boundray and write/read mask tests
removed : unsupported shader opcode

Change 55269 on 2002/10/04 by kevino@kevino_r400_win_marlboro

Added (uneeded) tex projection test case as well as get/set gradients test.
Updateds scene_data.h to have XYZW_ARGB_STQR4 in case every want Q and R.

Change 55200 on 2002/10/03 by csampayo@fl_csampayo_r400

Updated for multi-context

Change 55199 on 2002/10/03 by csampayo@fl_csampayo_r400

Added 4 new DMA swap tests and updated test_list and the test tracker

Change 55196 on 2002/10/03 by omesh@omesh_r400_linux_marlboro_only_devel

Fixed emulator testcases to scale color and alpha channels according to
the range allowed, as decided by the COLOR*_NUMBER register setting.

However, the emulator still doesn't seem to produce the right result
(Produces a non-interpolated black and gray triangle for the
NUMBER_UINTEGER and NUMBER_SINTEGER modes, respectively.

Change 55126 on 2002/10/03 by llefebvr@llefebvre_laptop_r400_emu

multiple fixes for memory exports.

Change 55093 on 2002/10/03 by abeaudin@abeaudin_r400_win_marlboro

fixed background pointer

Change 55079 on 2002/10/03 by vgoel@fl_vgoel2

added calculation for global ambient

Change 55069 on 2002/10/03 by mkelly@fl_mkelly_r400_win_laptop

Update runp to summarize PASS/FAIL
Begin to support block level envoking of runp

Change 55053 on 2002/10/03 by kevino@kevino_r400_win_marlboro

Moved EC exports to different ALU statements in shader program. Removed ecta waitforidle in cpp file.

Change 55048 on 2002/10/03 by kevino@kevino_r400_win_marlboro

Updated to use texture scale and offset in shader pipe, and use the TM.muladd function to
create appropriate data for the various fcomp values.

Change 55006 on 2002/10/03 by kevino@kevino_r400_win_marlboro

Get's computed LOD value and returns as reg value

Change 55005 on 2002/10/03 by kevino@kevino_r400_win_marlboro

Basic tmult (before scale/offset change)

Change 54970 on 2002/10/03 by mkelly@fl_mkelly_r400_win_laptop

Move p4_root determination out of get_sync() so it is always done

Change 54892 on 2002/10/02 by llefebvr@llefebvre_laptop_r400_emu

Fixed some memory export bugs.

Change 54879 on 2002/10/02 by csampayo@fl_csampayo_r400

Updated flush location

Change 54861 on 2002/10/02 by mkelly@fl_mkelly_r400_win_laptop

Perl script to run parameterized tests, first pass.
Checkpoint on sc template test to be used for stress testing.

Change 54839 on 2002/10/02 by kevino@kevino_r400_win_marlboro

Updated test case script to specify all 4 channel's format_comp's. Moved vertex buffer into a class in order to allow
multiple contextx later on if necessary,

Change 54830 on 2002/10/02 by ygiang@ygiang_r400_win_marlboro_p4

added:test for debug

Change 54825 on 2002/10/02 by omesh@omesh_r400_linux_marlboro_release

Added RB_SURFACE_SLICE programming that was missing for multisample
tests, as pointed out by Bill Lawless. Compiled test to make sure it
compiles and runs.

Change 54803 on 2002/10/02 by vgoel@fl_vgoel2

changed the test to non-tiled mode

Change 54802 on 2002/10/02 by csampayo@fl_csampayo_r400

Added 3 new VGT tests with negative index offsets and updated test_list and the test tracker

Change 54797 on 2002/10/02 by vgoel@fl_vgoel2

modified vertex shader and .cpp for perspective projection, and z-buffering

Change 54751 on 2002/10/02 by mkelly@fl_mkelly_r400_win_laptop

Changed indentation to more easily read a few lines

Change 54681 on 2002/10/01 by csampayo@fl_csampayo_r400

Updated for better test case coverage

Change 54672 on 2002/10/01 by vgoel@fl_vgoel2

changed the frame buffer size to 800 x 608

Change 54653 on 2002/10/01 by vgoel@fl_vgoel2

first cut in GI shaders

Change 54635 on 2002/10/01 by smoss@smoss_crayola_win

forgot one.

Change 54634 on 2002/10/01 by smoss@smoss_crayola_win

SU tests

Change 54625 on 2002/10/01 by mkelly@fl_mkelly_r400_win_laptop

Check point on SC template for stress testing...

Change 54600 on 2002/10/01 by omesh@omesh_r400_linux_marlboro_release

Added an explicit specification of "NUMBER_FLOAT" for floating point
color formats, as h/w is also going to change accordingly.
Changed the testnames 6 per test series to include the explicit
"number_float" in the testname. Also changed nightly regression lists to
the new testnames.

Change 54582 on 2002/10/01 by ygiang@ygiang_r400_win_marlboro_p4

added: shader dotx tests

Change 54579 on 2002/10/01 by mkelly@fl_mkelly_r400_win_laptop

Stipple permutations...

Change 54543 on 2002/10/01 by csampayo@fl_csampayo_r400

Renamed vertex shader export test

Change 54541 on 2002/10/01 by csampayo@fl_csampayo_r400

Renaming these tests

Change 54538 on 2002/10/01 by mkelly@fl_mkelly_r400_win_laptop

Complete coverage on stipple repeat count values 0 to 7.

Repeat Count Value	Test Name
0	r400sc_line_stipple_02
1	r400sc_line_stipple_01
2	r400sc_line_stipple_15
3	r400sc_poly_offset_09
4	r400sc_line_stipple_16

5	r400sc_line_strip_stipple_01
6	r400sc_line_stipple_17
7	r400sc_line_stipple_18

Change 54512 on 2002/10/01 by mkelly@fl_mkelly_r400_win_laptop

Varying combinations of Positive and Negative vtx window offsets

Change 54506 on 2002/10/01 by mkelly@fl_mkelly_r400_win_laptop

Negative vtx window offset in X and Y

Change 54488 on 2002/10/01 by mkelly@fl_mkelly_r400_win_laptop

Comment out SC_CHECK for double pixel hits until
primlib utility parser is updated to
latest sc_quad_pair_proc_out.dmp file

Change 54443 on 2002/09/30 by csampayo@fl_csampayo_r400

Added vertex export simple test

Change 54435 on 2002/09/30 by ygiang@ygiang_r400_win_marlboro_p4

added: new opcode tests

Change 54432 on 2002/09/30 by georgev@devel_georgevhw_r400_lin_marlboro

Changes to support enumerated types in random tests.

Change 54422 on 2002/09/30 by omesh@omesh_r400_linux_marlboro_release

Removed testcases that used gamma correction for components larger than
8 bits. If needed, will remove more gamma correction testcases later,
depending on whats supported. Removed a total of 6 testcases from each
test series = 12 tests removed.

Change 54395 on 2002/09/30 by omesh@omesh_r400_linux_marlboro_release

Removed invalid Color Formats (Any color components not 16 or 32 bits) using the NUMBER_FLOAT represention.
Removed a total of 13 testcases from each of 2 test series = 26
testcases removed.

Change 54357 on 2002/09/30 by ygiang@ygiang_r400_win_marlboro_p4

add: ieee boundary test
removed: unsupported sp tests

Change 54308 on 2002/09/30 by frivas@FL_FRivas

Update to "Matrix_Class" definition to allow code to compile under Linux.

Change 54302 on 2002/09/30 by mkelly@fl_mkelly_r400_win_laptop

Add two super tile tests which will only run standard case at this point

Change 54288 on 2002/09/30 by mkelly@fl_mkelly_r400_win_laptop

Overload VFD to handle specification of vertex buffer address constant in parameter
Usage example can be found in r400sanity_vfd_texture_sample_01.cpp

Change 54285 on 2002/09/30 by mkelly@fl_mkelly_r400_win_laptop

Change .sp filename to test name

Change 54275 on 2002/09/30 by vgoel@fl_vgoel2

changed the input file name to "r400_local_tonemapping.hdr"

Change 54274 on 2002/09/30 by vgoel@fl_vgoel2

changed the frame.dump() and size of frame

Change 54253 on 2002/09/29 by csampayo@fl_csampayo_r400

New VGT index offset tests

Change 54171 on 2002/09/27 by vgoel@fl_vgoel2

fixed several bugs in tone mapping shader and .cpp file.

Change 54139 on 2002/09/27 by ygiang@ygiang_r400_win_marlboro_p4

added: 3 constant alu register muladd operation

Change 54111 on 2002/09/27 by ygiang@ygiang_r400_win_marlboro_p4

fixed: # of args in the set opcode to matched new instruction in the assembler

Change 53973 on 2002/09/26 by ygiang@ygiang_r400_win_marlboro_p4

fixed: number of args in shader tests cause changes from the parser

Change 53964 on 2002/09/26 by omesh@omesh_r400_linux_marlboro_only_devel

Changed atleast 2 tests to match Kevin Ryan's new memory dump routine interface. (Added an extra parameter). The tests now continue to run, but I haven't (re)verified the output.

Change 53954 on 2002/09/26 by smoss@smoss_crayola_win

SU test

Change 53953 on 2002/09/26 by smoss@smoss_crayola_win

SU tests

Change 53941 on 2002/09/26 by hwise@fl_hwise_r400_win

Adding test for debugging Primlib memory alloc overlap problem. (The Index Buffer is overlapping the Ring Buffer)

Change 53940 on 2002/09/26 by ygiang@ygiang_r400_test_marlboro

added: more test

Change 53938 on 2002/09/26 by csampayo@fl_csampayo2_r400

Updated flush location

Change 53931 on 2002/09/26 by kevino@kevino_r400_win_marlboro

Explicetly set texture dimension

Change 53807 on 2002/09/26 by kevino@kevino_r400_win_marlboro

Test case files

Change 53801 on 2002/09/26 by vgoel@fl_vgoel2

updated adaptive tessellation test

Change 53780 on 2002/09/26 by csampayo@fl_csampayo2_r400

Uncommented tests that should run now:
r400vgt_hos_simple_linear_PNT_discrete_01
r400vgt_hos_TPatch_01
r400vgt_hos_TPatch_02

Change 53770 on 2002/09/26 by kevino@kevino_r400_win_marlboro

Get rid of savemipchain ppm's

Change 53738 on 2002/09/25 by csampayo@fl_csampayo2_r400

Commented out test not running: r400vgt_event_handling_01

Change 53713 on 2002/09/25 by omesh@omesh_r400_linux_marlboro_only_devel

Added 20 testcases which cover most cases of testing fragment based multisampling. These tests are geared towards probing the fragment buffer and color cache with most combinations of CMASK values and Max Fragment coverage per pixel regressed. Another 4 testcases test the fragment buffer overflow condition and 2 random tests randomize color.
I have yet to add a series of testcases to test uninitialized samples.
Also, I have run these 20 tests on Linux but could not see the output as DumpView is not ready to display multisampled images yet.

Change 53659 on 2002/09/25 by kevino@kevino_r400_win_marlboro

Just wanna look at them with xv then delete them

Change 53637 on 2002/09/25 by kevino@kevino_r400_win_marlboro

1st step of triangle_vertex_buffer replacement.

Change 53607 on 2002/09/25 by mkelly@fl_mkelly_r400_win_laptop

Poly offset / scale tests

Change 53579 on 2002/09/25 by hartogs@fl_hartogs

Modifications for to fix quads and line-loops for multi-prim reset functionality.

Change 53571 on 2002/09/25 by omesh@omesh_r400_linux_marlboro_only_devel

Fixed some syntax errors, and some warnings and verified that the test runs. Could not verify the output image, as DumpView isn't ready to display fragment based multisampled images yet.

Change 53560 on 2002/09/25 by kevino@kevino_r400_win_marlboro

Added getcomplod test

Change 53535 on 2002/09/25 by kevino@kevino_r400_win_marlboro

Added line that outputs texture base offset

Change 53532 on 2002/09/25 by csampayo@fl_csampayo_r400

Commented out test hanging the emulator: r400vgt_suppress_eop_05

Change 53502 on 2002/09/25 by mkelly@fl_mkelly_r400_win_laptop

Log potential bug

Change 53499 on 2002/09/25 by mkelly@fl_mkelly_r400_win_laptop

Poly Offset combinations...

Change 53470 on 2002/09/24 by georgev@devel_georgevhw_r400_lin_marlboro

Done Fixed some bad tests.

Change 53451 on 2002/09/24 by csampayo@fl_csampayo_r400

Added new VGT test. Updated test_list and test tracker

Change 53442 on 2002/09/24 by ygiang@ygiang_r400_win_marlboro_p4

fixed: random test case, using large random number now

Change 53414 on 2002/09/24 by omesh@omesh_r400_linux_marlboro_release

Added the major case of the multisample tests. Need to split this up into several testcases to simplify possible debugging. It uses Kevin Ryan's new COLOR_SURFACE class that supports multisampled Color Buffer 0, including the Dump.
I have not yet run this test or tested the result (DumpView may not be ready to display multisampled Color Buffers yet).
Need to add: 1) Random testcases to this. 2) Corner cases: Fragment buffer overflow and unassigned samples test cases.

Change 53409 on 2002/09/24 by ygiang@ygiang_r400_test_marlboro

added:interpolator test

Change 53385 on 2002/09/24 by mkelly@fl_mkelly_r400_win_laptop

Polymode poly offset, front/back face, MSAA, sample control permutations

Change 53342 on 2002/09/24 by frivas@FL_FRivas

Finished Line Patch test. Renders 4 line patches, each with 4 control points. Tessellation varies 1-14.5 and reuse varies 4-16.

Change 53281 on 2002/09/24 by mkelly@fl_mkelly_r400_win_laptop

Temporarily remove test which changes out-of-context state

Change 53200 on 2002/09/23 by csampayo@fl_csampayo_r400

Added new VGT index reset test. Updated test_list and test tracker

Change 53185 on 2002/09/23 by ygiang@ygiang_r400_test_marlboro

changed: runtest to bruntest

Change 53183 on 2002/09/23 by ygiang@ygiang_r400_test_marlboro

added: sp regression test

Change 53182 on 2002/09/23 by csampayo@fl_csampayo2_r400

Changed buffers used to 1 and Inserted flushes between packets

Change 53154 on 2002/09/23 by ygiang@ygiang_r400_test_marlboro

update

Change 53116 on 2002/09/23 by ygiang@ygiang_r400_test_marlboro

sp mini regress script

Change 53107 on 2002/09/23 by mkelly@fl_mkelly_r400_win_laptop

Include history file with PASS/FAIL for auto depot submit

Change 53025 on 2002/09/23 by ctaylor@fl_ctaylor_r400_dtwin_marlboro

Modified Viz Query test to make begin event occur after the renderstate definition since the event uses renderstate.

Change 53019 on 2002/09/23 by mkelly@fl_mkelly_r400_win_laptop

Poly Offset, 8 MSAA, HOS

Change 53015 on 2002/09/23 by vgoel@fl_vgoel2

changed tessellation level to 14.9 to match with previous version of test

Change 52934 on 2002/09/21 by ygiang@ygiang_r400_win_marlboro_p4

moded: testcases

Change 52875 on 2002/09/20 by csampayo@fl_csampayo_r400

Updated number of buffers used

Change 52866 on 2002/09/20 by csampayo@fl_csampayo_r400

Added 2 new VGT tests and updated test_list and the test tracker accordingly

Change 52808 on 2002/09/20 by omesh@omesh_r400_linux_marlboro_release

Modified linear tests to also use the old FRAMEBUFFER_MEMORY class with
the FMT_* enumerations for SurfaceFormat. Also added explicit flags to
Fill and Dump routines in Primlib to tell it if the framebuffer is tiled
or not, so when the image is brought up on DumpView, it reads the right
information encoded in the header and brings up the image in the right
mode.

Change 52799 on 2002/09/20 by ygiang@ygiang_r400_win_marlboro_p4

added: test name in make file

Change 52798 on 2002/09/20 by kryan@kryan_r400_win_marlboro

MEMORY_AREA

- Begin work on new Dump() function that takes an enumerated type

for the type of Surface being dumped.

- Add enumerated types for surface types: CB0,CB1,CB2,CB3,ZB,TB

memory_utility.cpp

- Changed Write_To_Memory(TEXTURE_BUFFER) to use format specified

in TEXTURE_BUFFER when calling other functions.

TEXTURE_BUFFER

- Modified code to use SurfaceFormat information to calculate

bits per sample rather than bps value passed in through functions.

This also allowed the elimination of bps parameter to all the

functions.

- Added code for all functions including constructor to have the

ability to use uint8* in addition to uint32* for storing/

manipulating texture data.

- Implemented code to store textures as bytes rather than uint32s.

texture_utility.h/.cpp

- Added comments/formatting.

- Added uint8* version of Load_Texture_And_Write_To_Memory(uint8*)

SURFACE

- Add depth member and access functions

- Add number of samples member and access functions

- For dump file generation:

- Add Add_Define(), Add_Comment(), Add_Property_Defines() functions

- Add Get_Surface_Type() function to return string for Surface Type.

Default to "GENERIC".

PIXEL_SURFACE

- Add depth member and access functions

- Added Fill_Solid(color) function that will use default pitch,height

values of surface.

COLOR_SURFACE

- Work on class with specific Add_Property_Defines() function

results_diff.pl

- Modify .diff_e target through diff function to ignore

all header information in dumpfiles for Marlboro and

Orlando sites.

REGISTER_WRITE_MANAGER

- Add code to initialize pl3d::LibType with value based upon mode of

(PM4 or PIO) of constructor call.

test_lib/src/chip/perf/chip/ various tests

- Updated various tests using the PIXEL_SURFACE class to add the

now required depth parameter.

- Also removed the pitch,height parameters from the

PIXEL_SURFACE::Fill_Solid(color) function since they default to

use the dimensions of the surface itself.

test_lib/src/chip/perf/primlib_tex_tri

- Update primlib_tex_tri.cpp test to reflect new Dumpfile format and

content for multisample ColorSurface0.

Change 52797 on 2002/09/20 by ygiang@ygiang_r400_win_marlboro

added: default alpha fetch value

Change 52786 on 2002/09/20 by mkelly@fl_mkelly_r400_win_laptop

Moved former bug into regular testing regression.

Change 52783 on 2002/09/20 by mkelly@fl_mkelly_r400_win_laptop

Move test to SC for baryc validation

Change 52725 on 2002/09/20 by kevino@kevino_r400_win_marlboro

Some more textures (1D)

Change 52724 on 2002/09/20 by kevino@kevino_r400_win_marlboro

Assortment of files I forgot to check in
gen_rg- generates .rg files (w/o header!) by looking for test case defines in the
.cpp and it's includes
aniso, clamp test cases.

Change 52723 on 2002/09/20 by kevino@kevino_r400_win_marlboro

Updated alu_const_array stuff
Added basics for functions to put tscale and toffstes into alu const regs 4-7, 8-11.
Shader pipe needs to multiply these. Used to shift all the various number formats back into 0-1 range
for color export.

Change 52706 on 2002/09/20 by mkelly@fl_mkelly_r400_win_laptop

Update to clip lines inside HW boundary of +-8k

Change 52701 on 2002/09/20 by mkelly@fl_mkelly_r400_win_laptop

Add test to regression since bug is fixed, re: change 49961, last item

Change 52672 on 2002/09/19 by omesh@omesh_r400_linux_marlboro_release

Switched back to using the old FRAMEBUFFER_MEMORY class for Primlib fill
and dump routines and they seem to work for the most part except for
about 8 modes that don't dump and another 3 that dump incorrect images.
I have filed Bugzilla bugs for Kevin Ryan to look at.
Also, fixed a bug with the tests to program it for tiled mode of
operation. Still haven't validated the output images of these tests due
to DumpView's automatic nature of scaling all color channels to full
precision even for modes in which other channels do not exit (example
COLOR_8 or FMT_8 which looks a little wierd on DumpView).

Change 52648 on 2002/09/19 by csampayo@fl_csampayo_r400

Added 1 new VGT event handling test and updated the test_list and test tracker accordingly

Change 52630 on 2002/09/19 by abeaudin@abeaudin_r400_win_marlboro

changed program and vertex shaderto get a solid red and a solid green triangle

Change 52594 on 2002/09/19 by georgev@devel_georgevhw_r400_lin_marlboro

Ooops. Fixed dma_32 test.

Change 52592 on 2002/09/19 by jhoule@jhoule_r400_win_marlboro

tp_uber_test:
- updated programs to have cleaner names
- removed RGB from vertex arrays
- now sets DIM field for loads, since the TEXTURE_MANAGER requires that
- honors DIM field for the same reason (defaults to 2D, though)

Change 52583 on 2002/09/19 by vgoel@fl_vgoel2

changed z-buffer
changed frame.dump()

Change 52579 on 2002/09/19 by vgoel@fl_vgoel2

changed z-buffer
changed frame.dump()

Change 52574 on 2002/09/19 by mkelly@fl_mkelly_r400_win_laptop

Update to properly order draws/flushes/reads/ and writes...

Change 52569 on 2002/09/19 by vgoel@fl_vgoel2

changed z-buffer
changed frame.dump()
added compute_dma_size()

Change 52560 on 2002/09/19 by vgoel@fl_vgoel2

changed z-buffer
changed frame.dump()
added compute_dma_size()

Change 52556 on 2002/09/19 by vgoel@fl_vgoel2

changed to display only one T-patch object
changed z-buffer
changed frame.dump() function
added compute_dma_size

Change 52541 on 2002/09/19 by mkelly@fl_mkelly_r400_win_laptop

Enable GB clipping, adjust W to prevent lines falling exactly on HW
boundary and width making them extend beyond HW boundary.

Change 52535 on 2002/09/19 by vgoel@fl_vgoel2

changed z-buffer and frame.dump()

Change 52533 on 2002/09/19 by georgev@devel_georgevhw_r400_lin_marlboro

No changes.

Change 52506 on 2002/09/19 by mkelly@fl_mkelly_r400_win_laptop

Cycle SC_SAMPLE_CNTL permutations with all primtypes
Cycle MSAA, 192 packets

Change 52498 on 2002/09/19 by mkelly@fl_mkelly_r400_win_laptop

Another example for bugzilla 401

Change 52486 on 2002/09/19 by kevino@kevino_r400_win_marlboro

1D texture that alternates color and black/grey

Change 52482 on 2002/09/19 by mkelly@fl_mkelly_r400_win_laptop

Primitive should be gouraud shaded, it is flat shaded.
Related to PARAM_GEN = true, CENTERS_ONLY, and SAMPLE_CENTER
If PARAM_GEN = false, gouraud shading works as expected.

Change 52464 on 2002/09/19 by mkelly@fl_mkelly_r400_win_laptop

Changed PA to TOTAL

Change 52463 on 2002/09/19 by mkelly@fl_mkelly_r400_win_laptop

Rename MISC to PERF

Change 52442 on 2002/09/18 by ygiang@ygiang_r400_win_marlboro_p4

fixed: test case random, now using large random number system and not TG

Change 52439 on 2002/09/18 by georgev@devel_georgevhw_r400_lin_marlboro

Fixed 2 tests.

Change 52429 on 2002/09/18 by csampayo@fl_csampayo_r400

Added 5 new VGT prim tests and updated test_list and test tracker accordingly

Change 52417 on 2002/09/18 by ygiang@ygiang_r400_test_marlboro

changed: abs code, it's $ instead of &

Change 52355 on 2002/09/18 by csampayo@fl_csampayo_r400

Updated so that it runs with latest Primlib

Change 52354 on 2002/09/18 by kevino@kevino_r400_win_marlboro

Added 1D texture clamp cases based on 2D cases.

Change 52340 on 2002/09/18 by mkelly@fl_mkelly_r400_win_laptop

Re-enable r400sc_point_list_06 to regression now it mysteriously works.

Change 52336 on 2002/09/18 by mkelly@fl_mkelly_r400_win_laptop

Enable regression of fixed bugs

Change 52300 on 2002/09/18 by mkelly@fl_mkelly_r400_win_laptop

Cycle primitive types through all MSAA and JSS modes.

Change 52286 on 2002/09/18 by kevino@kevino_r400_win_marlboro

Added GetBorderColoFraction test
updated lod test to hit instr bias, const bias, comp, and reg_lod.

Change 52259 on 2002/09/18 by frivas@FL_FRivas

Update

Change 52256 on 2002/09/18 by frivas@FL_FRivas

Update to matrix code to allow it to work under Linux.

Change 52216 on 2002/09/18 by mkelly@fl_mkelly_r400_win_laptop

Add flush between packets but assert still occurs...

Change 52204 on 2002/09/18 by mkelly@fl_mkelly_r400_win_laptop

RB asserts when polygon with MSAA = 0,1,2,3 followed
by point list MSAA = 5. If polygon MSAA = 5 or 7 the
test completes as expected.

$/r400/devel/test_list/src/chip/gfx/pa/bugs

make r400sc_sp_sample_cntl_08_bug.emu

```
Assertion failed: (coverage&errmask) == 0, file ../../../../emu_lib/model/gfx/rb
/rb_color_model.cpp, line 334
make[2]: [run_emu] Error 3 (ignored)
```

Change 52180 on 2002/09/17 by csampayo@fl_csampayo_r400

Added 4 new VGT prim tests and updated test_list and the test tracker accordingly.

Change 52168 on 2002/09/17 by omesh@omesh_r400_linux_marlboro_release

Changed size of display framebuffer and triangles so that DumpView can
display the images faster, on Alicia's request.

Change 52133 on 2002/09/17 by georgev@devel_georgevhw_r400_lin_marlboro

Regression fixes.

Change 52131 on 2002/09/17 by ygiang@ygiang_r400_win_marlboro_p4

added: rb swap color channels register

Change 52045 on 2002/09/17 by kevino@kevino_r400_win_marlboro

tcfux for texture offsets (sample shift)

Change 52041 on 2002/09/17 by omesh@omesh_r400_linux_marlboro_release

Removed Color Format modes (non blendable) that don't support Color Masking, from the random test case. Split 64 bit LOG_PARAMETER call into 2 x 32 bit LOG_PARAMETER calls so that the linker on Windows doesn't complain (On Linux this seems to work with no problem)

Change 52022 on 2002/09/17 by kryan@kryan_r400_win_marlboro

Modified top-level catch block of test to replace throw statement

with a return 1; so that the test will exit gracefully without a

dialog box if an error occurs.

Change 52004 on 2002/09/17 by mkelly@fl_mkelly_r400_win_laptop

Fix percent calculations for cp and rbbm

Change 51992 on 2002/09/17 by frivas@FL_FRivas

Added a multi-prim HOS test to the list.

Change 51986 on 2002/09/17 by mkelly@fl_mkelly_r400_win_laptop

Update comment...

Change 51984 on 2002/09/17 by mkelly@fl_mkelly_r400_win_laptop

Publish Orlando full regression results to //depot/r400/web when "-p" option is true.

Change 51963 on 2002/09/17 by mkelly@fl_mkelly_r400_win_laptop

regress_r400

Change 51911 on 2002/09/16 by ygiang@ygiang_r400_win_marlboro_p4

fixed: image for tests

Change 51897 on 2002/09/16 by csampayo@fl_csampayo_r400

Added 3 new VGT prim tests and updated test_list and the test tracker accordingly.

Change 51874 on 2002/09/16 by georgev@devel_georgevhw_r400_lin_marlboro

Fixed bad screen size bug.

Change 51873 on 2002/09/16 by omesh@omesh_r400_linux_marlboro_release

Added another 216 test cases. Most of them don't display correctly on DumpView and many don't dump correctly with
Primlib yet. The output image is yet to be verified for these tests.

Change 51868 on 2002/09/16 by ygiang@ygiang_r400_win_marlboro_p4

removed: position clamping from vertex shader tests

Change 51851 on 2002/09/16 by ygiang@ygiang_r400_win_marlboro_p4

added: temp golden images cause golden does not support test cases yet.

Change 51838 on 2002/09/16 by ygiang@ygiang_r400_test_marlboro

added: simple passthrough test for debug.

Change 51812 on 2002/09/16 by ygiang@ygiang_r400_win_marlboro_p4

fixed: tri size

Change 51763 on 2002/09/16 by mkelly@fl_mkelly_r400_win_laptop

More extensive back face bit checks

Change 51753 on 2002/09/16 by frivas@FL_FRivas

Addition to multi prim test to allow it to work with quads.

Change 51752 on 2002/09/16 by frivas@FL_FRivas

Update. This test now alternates between drawing lines, triangles and quads (PNL, PNT, PNQ). Cubic position and quadratic normals are used.

Change 51748 on 2002/09/16 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 51747 on 2002/09/16 by mkelly@fl_mkelly_r400_win_laptop

Check back face bit carries from sc_sp to shader export to RB, looking good...

Change 51723 on 2002/09/16 by kevino@kevino_r400_win_marlboro

Don't try to delete ptr if null

Change 51539 on 2002/09/13 by georgev@devel_georgevhw_r400_lin_marlboro

Changes for CP.

Change 51527 on 2002/09/13 by pmitchel@pmitchel_entire_depot_win

only delete ptr if ptr != NULL

Change 51516 on 2002/09/13 by ygiang@ygiang_r400_win_marlboro_p4

added: override Shader Instruction Loading

Change 51504 on 2002/09/13 by kevino@kevino_r400_win_marlboro

Added cubic testcase and support.
Fixed deallocate to check if the ptr is null before it tries to delete it and only deletes it if it is not.

Change 51490 on 2002/09/13 by csampayo@fl_csampayo_lt_r400

Added to test_list and updated status for test:
r400vgt_multi_prim_reset_index_all_01

Change 51483 on 2002/09/13 by csampayo@fl_csampayo_r400

Adding new VGT mul-prim index reset test

Change 51468 on 2002/09/13 by ygiang@ygiang_r400_win_marlboro

Changed: buffers and triangles size for hardware testing

Change 51451 on 2002/09/13 by kevino@kevino_r400_win_marlboro

needed for tp_simple_02/complex_shader testcase

Change 51446 on 2002/09/13 by mkelly@fl_mkelly_r400_win_laptop

Update for Randy to try

Change 51428 on 2002/09/13 by omesh@omesh_r400_linux_marlboro_release

Added code for random selection of Color and Surface format. Currently hardcoded the Color/Surface format, to trap a specific dump that doesn't work in
Primlib (Also filed a Bugzilla bug (Bug ID 371) on this.

Change 51386 on 2002/09/13 by csampayo@fl_csampayo_r400

Update for new DMA SIZE calculation

Change 51354 on 2002/09/13 by ygiang@ygiang_r400_win_marlboro_p4

changed: tri-size for hardware testing

Change 51317 on 2002/09/13 by mkelly@fl_mkelly_r400_win_laptop

Enable GB clipping to ensure verts don't extend beyond HW boundary

Change 51311 on 2002/09/13 by mkelly@fl_mkelly_r400_win_laptop

Add CP and RBBM blocks to parse, search, make, and reporting routines

Change 51272 on 2002/09/12 by csampayo@fl_csampayo2_r400

Commented out tests not working

Change 51261 on 2002/09/12 by csampayo@fl_csampayo_r400

Updated for new dma size calculation

Change 51215 on 2002/09/12 by omesh@omesh_r400_linux_marlboro_release

Checking in the 14 multiwrite (multiple render target) tests.
The 3 and 4 exports->buffers tests seem to timeout on the emulator and the 2 exports->buffers tests seem to produce the
wrong result. I will file a Bugzilla bug.

Change 51201 on 2002/09/12 by csampayo@fl_csampayo_r400

1. Added 5 new VGT provoking vtx tests
2. Updated test_list and test tracker

Change 51137 on 2002/09/12 by vgoel@fl_vgoel2

changed the input primitive type to 3D point for adaptive tessellation and added
draw_command.Calculate_Dma_Size (render_state1);

Change 51128 on 2002/09/12 by frivas@FL_FRivas

Minor update to all HOS tests that adds the line "draw_command.Calculate_Dma_Size(render_state1);"

Change 51080 on 2002/09/12 by mkelly@fl_mkelly_r400_win_laptop

SC screen XY output tests

Change 50987 on 2002/09/12 by ygiang@ygiang_r400_win_marlboro_p4

changed: triangles size for hardware testing

Change 50984 on 2002/09/12 by mkelly@fl_mkelly_r400_win_laptop

Finalized SC screen XY / centers / centroids initial test, correct
results are obtained through the RB when in hardware accurate mode
otherwise, resort to sp_sx.dmp to confirm

Change 50933 on 2002/09/11 by kmahler@kmahler_r400_win_devel_views

Fixed setting of DMA_SIZE field in the DRAW_INDX packet for major mode 1.

Below is how to program Primlib to automatically have the RENDER_ENGINE set the correct VGT_DMA_SIZE in the DRAW_INDX packet.

When the test initializes the DRAW_COMMAND....

```
//-----------------------------------------------------------------------------------------
// Set up for Major mode 1 (NORMAL_NO_VERTEX_REUSE)
//
// Calculate_Dma_Size() should only be invoked when major mode 1 is enabled and
// after setting the number of indices in the draw_command.
//-----------------------------------------------------------------------------------------
draw_command.Set_Major_Mode(
DRAW_COMMAND::NORMAL_NO_VERTEX_REUSE);
  .
  .
  .

draw_command.Set_Number_Indices ( Num_Indx );
draw_command.Calculate_Dma_Size(render_state);

  .
  .
  .
```

******* NOTE
****

There are several SQ and VGT tests that set major mode 1. These tests may have to be modified in order to work.

```
sq\r400sq_bell.cpp(530):
sq\r400sq_plane.cpp(513):
sq\r400sq_plane_env_map.cpp(563):
sq\r400sq_shell.cpp(530):
sq\r400sq_simple_obj.cpp(515):
sq\r400sq_simple_obj_env_map.cpp(556):
sq\r400sq_sphere.cpp(531):
sq\r400sq_sphere_env_map.cpp(564):
sq\r400sq_sphere_less.cpp(530):
vgt\r400_local_tonemapping.cpp(1110):
vgt\r400_stereo_vision.cpp(861):
vgt\r400vgt_hos_LINE_cp.cpp(445):
vgt\r400vgt_hos_LPatch_01.cpp(445):
vgt\r400vgt_hos_PNL_cp_ln_cont_no_projection_01.cpp(798):
vgt\r400vgt_hos_PNL_lp_ln_cont_no_projection_01.cpp(795):
vgt\r400vgt_hos_PNQ_cp_qn_cont_light_texture_01.cpp(808):
vgt\r400vgt_hos_PNQ_cp_qn_cont_light_texture_02.cpp(808):
vgt\r400vgt_hos_PNQ_cp_qn_cont_no_projection_01.cpp(807):
vgt\r400vgt_hos_PNQ_lp_cont_no_projection_01.cpp(791):
vgt\r400vgt_hos_PNTQL_cp_qn_cont_stress_01.cpp(670):
vgt\r400vgt_hos_PNT_adaptive.cpp(893):
vgt\r400vgt_hos_PNT_cont_cp_qn_complex_01.cpp(856):
vgt\r400vgt_hos_PNT_cont_cp_qn_precision_01.cpp(768):
vgt\r400vgt_hos_PNT_cont_cp_qn_precision_02.cpp(764):
vgt\r400vgt_hos_PNT_cp_qn_cont_light_texture_01.cpp(799):
vgt\r400vgt_hos_PNT_cp_qn_cont_light_texture_02.cpp(801):
vgt\r400vgt_hos_PNT_cp_qn_cont_light_texture_03.cpp(801):
vgt\r400vgt_hos_PNT_cp_qn_cont_moving_normals_01.cpp(758):
vgt\r400vgt_hos_PNT_cp_qn_cont_no_projection_01.cpp(782):
vgt\r400vgt_hos_PNT_cp_qn_disc_14_04_lit_tex_proj_01.cpp(677):
vgt\r400vgt_hos_PNT_disc_cp_qn_complex_01.cpp(855):
vgt\r400vgt_hos_PNT_disc_cp_qn_light_texture_01.cpp(799):
vgt\r400vgt_hos_PNT_disc_cp_qn_no_projection_01.cpp(782):
vgt\r400vgt_hos_PNT_disc_cp_qn_precision_01.cpp(768):
vgt\r400vgt_hos_PNT_disc_cp_qn_precision_02.cpp(763):
vgt\r400vgt_hos_PNT_edge_detection_01.cpp(992):
vgt\r400vgt_hos_PNT_lp_cont_no_projection_01.cpp(784):
vgt\r400vgt_hos_RPatch_02.cpp(639):
vgt\r400vgt_hos_RPatch_cp_02.cpp(638):
vgt\r400vgt_hos_RPatch_lp_02.cpp(637):
vgt\r400vgt_hos_TPatch_01.cpp(696):
vgt\r400vgt_hos_TPatch_02.cpp(696):
vgt\r400vgt_hos_TRI_cp.cpp(563):
vgt\r400vgt_hos_auto_index_line_list_01.cpp(427):
vgt\r400vgt_hos_auto_index_quad_list_01.cpp(452):
vgt\r400vgt_hos_auto_index_triangle_list_01.cpp(468):
```

```
vgt\r400vgt_hos_cubic_pos_pnt_discrete_01.cpp(494):
vgt\r400vgt_hos_simple_linear_PNT_discrete_01.cpp(490):
vgt\r400vgt_pass_thru_all_prims_01.cpp(385):
```

Change 50932 on 2002/09/11 by fhsien@fhsien_r400_linux_marlboro

Update for smaller images

Change 50820 on 2002/09/11 by mkelly@fl_mkelly_r400_win_laptop

Check point, test is almost complete...

Change 50815 on 2002/09/11 by ctaylor@fl_ctaylor_r400_dtwin_marlboro

Moved 6-Sample MSAA Sample #2 location from ULC-rel 2,5 to 1,5 to alleviate degen tri in texture lod computations. (HW and EMU and fixed 1 test).

Change 50794 on 2002/09/11 by fhsien@fhsien_r400_linux_marlboro

Update for smaller images

Change 50707 on 2002/09/11 by mkelly@fl_mkelly_r400_win_laptop

Simple triangle, polymode back face tri fill

Change 50701 on 2002/09/11 by mkelly@fl_mkelly_r400_win_laptop

Add sq dumps to regression house keeping

Change 50676 on 2002/09/11 by omesh@omesh_r400_linux_marlboro_release

Added code to program SurfaceFormat registers as well as detemine the
number of bytes per pixel, based on the ColorFormat, instead of having
each test case specify them. Most color formats still don't display
correctly with DumpView (bug filed) and the SurfaceFormat class doesn't
seem to dump any color format correctly, as yet.
Also added code for random test cases.

Change 50645 on 2002/09/10 by kevino@kevino_r400_win_marlboro

Worked on reglod test (but use_reg_lod is a bool in tfetch_instruction.cpp and an enum in tinstr.h)
Got offset tests working and added pos and neg cases to cover the whole range.

Change 50623 on 2002/09/10 by smoss@smoss_crayola_win

su tests

Change 50588 on 2002/09/10 by kevino@kevino_r400_win_marlboro

Fixed border size test- TM now sets texture size correctly (-1 per side for border)
Added TFetchInst override of mip, minmag, and aniso in const

Change 50581 on 2002/09/10 by csampayo@fl_csampayo2_r400

Commented out tests that hang regression.

Change 50569 on 2002/09/10 by mkelly@fl_mkelly_r400_win_laptop

Fix name spelling

Change 50497 on 2002/09/10 by mkelly@fl_mkelly_r400_win_laptop

Narrowed down the bug to polymode tri fill in the SC

Change 50491 on 2002/09/10 by hwise@fl_hwise_r400_win

Updating registry files in pa_regress to choose the CP Microengine
configuration rather than the 'C' implementation

Change 50486 on 2002/09/10 by mkelly@fl_mkelly_r400_win_laptop

log a potential bug

Change 50454 on 2002/09/10 by jhoule@jhoule_r400_win_marlboro

Added support for BORDER_SIZE specification.
Added some textures and test files.
Updated formats.tst, and the shader programs.

Change 50428 on 2002/09/10 by mkelly@fl_mkelly_r400_win_laptop

For SC sample control #2

Change 50427 on 2002/09/10 by mkelly@fl_mkelly_r400_win_laptop

SC sample control centers, centroids and Screen XY

Change 50408 on 2002/09/10 by mkelly@fl_mkelly_r400_win_laptop

Check parameter 0 for SC center/centroid sampling control

Change 50375 on 2002/09/09 by fhsien@fhsien_r400_linux_marlboro

Update with smaller images

Change 50347 on 2002/09/09 by csampayo@fl_csampayo_r400

1. Added SU multi-context test
2. Updated test_list and test tracker accordingly

Change 50335 on 2002/09/09 by omesh@omesh_r400_linux_marlboro_release

Added 28 more tests. Either the .dump() routines don't dump correctly,
or DumpView does not display 16_16_16_16 and 8_8_8_8 formats correctly.
Seems more likely that the dump routines don't work correctly.

Change 50325 on 2002/09/09 by ygiang@ygiang_r400_test_marlboro

changed: temp vertex buffer to uint32 for hex format

Change 50272 on 2002/09/09 by smoss@smoss_crayola_linux_orl_regress

New VGT file

Change 50250 on 2002/09/09 by mkelly@fl_mkelly_r400_win_laptop

Add Scott's new dump to regression
Rename test

Change 50212 on 2002/09/09 by mkelly@fl_mkelly_r400_win_laptop

Added two new sc dumps to regress compare list

Change 50206 on 2002/09/09 by smoss@smoss_crayola_win

su tests

Change 50196 on 2002/09/09 by kevino@kevino_r400_win_marlboro

Added 1st cut at RegLod test.

Change 50187 on 2002/09/09 by ygiang@ygiang_r400_win_marlboro_p4

changed: test case

Change 50176 on 2002/09/09 by kevino@kevino_r400_win_marlboro

Added tcdenorm non-power2 case.

Change 50172 on 2002/09/09 by kevino@kevino_r400_win_marlboro

Added TFetchInst dst and src sel operations as well as tx_coord_denorm testcase

Change 50168 on 2002/09/09 by omesh@omesh_r400_linux_marlboro_release

Changed the "Range" type of random number generator to "RangeF" so it

returns the right kind (float) of data structure. Default behavior was
changed by George a while ago, which was resulting in 0 valued returns.

Change 50143 on 2002/09/09 by omesh@omesh_r400_linux_marlboro_release

Changed the "Range" type of random number generator to "RangeF" so it
returns the right kind (float) of data structure. Default behavior was
recently changed by George, which was resulting in 0 valued returns.
I'll change this soon for all earlier random tests as well.

Change 50121 on 2002/09/09 by mkelly@fl_mkelly_r400_win_laptop

Fix tests to update all GB RAM registers between state change

Change 50050 on 2002/09/06 by llefebvr@llefebvre_laptop_r400_emu

Fixed SWAP error that caused missmatches with gold images.
Added count reset capability in the SQ for multipass.
Added better comments for the PARAM_SHADE register field.
Fixed some HW accurate bugs.
Added new MOVA write back to GPRs feature.

Change 50033 on 2002/09/06 by kevino@kevino_r400_win_marlboro

tcfunc file for endian swap tests

Change 50031 on 2002/09/06 by kevino@kevino_r400_win_marlboro

Added trijuice

Change 50014 on 2002/09/06 by mkelly@fl_mkelly_r400_win_laptop

Initial check of sc sample control for centers and centroids in the sc_sp

Change 50001 on 2002/09/06 by mkelly@fl_mkelly_r400_win_laptop

Change ScDmp level to 3

Change 49995 on 2002/09/06 by mkelly@fl_mkelly_r400_win_laptop

Stipple tests where lines are small, and repeat count and
pattern are changed between state

Change 49977 on 2002/09/06 by abeaudin@abeaudin_r400_win_marlboro

more golden images

Change 49975 on 2002/09/06 by mkelly@fl_mkelly_r400_win_laptop

BRESENHAM CONTROL verified to match R200 in following tests.
For W9X, use 0x55, for W2K use 0x53

Change 49968 on 2002/09/06 by abeaudin@abeaudin_r400_win_marlboro

more new goldens

Change 49938 on 2002/09/06 by csampayo@fl_csampayo_r400

Updated Viz Query controls. Updated test status

Change 49937 on 2002/09/06 by frivas@FL_FRivas

this test has a lot of bugs. It will be replaced soon.

Change 49930 on 2002/09/06 by kevino@kevino_r400_win_marlboro

Added dstsel, miprange, and bordercolor tests and fmt565 files.
RF (repeating fraction) tests are there, but not yet working.

Change 49901 on 2002/09/06 by mkelly@fl_mkelly_r400_win_laptop

Missed checking in this one yesterday...

Change 49899 on 2002/09/06 by mkelly@fl_mkelly_r400_win_laptop

First pass of complex pinwheel

Change 49880 on 2002/09/05 by omesh@omesh_r400_linux_marlboro_release

Added some more testcases, fixed MAKE_VISUAL testcases (testcases involving *only* the alpha channel, which are
color blended with Red in the end, to bring out their visual result).
I have visually verified some more testcases on the emulator, including the 3 newly added ones.

Change 49844 on 2002/09/05 by omesh@omesh_r400_linux_marlboro_release

Explicitly set the COLOR*_SWAP field for each of the Color Buffers in the RB_COLOR_INFO register.
The emulator's default setting for all 4 Color Buffers was not the same, which was resulting in swapping of
the Red and Blue channels.
Also added more testcases, including random testcases and added post render Destination Alpha blend, to bring
out the effects of the final Destination Alpha (using Red color) if needed.
I haven't yet checked all the visual results of all the testcases, but I think they should be fine.

Change 49837 on 2002/09/05 by smoss@smoss_crayola_win

housekeeping

Change 49822 on 2002/09/05 by csampayo@fl_csampayo_r400

1. Added 1 new VGT test for event handling
2. Updated test_list and test tracker accordingly

Change 49804 on 2002/09/05 by frivas@FL_FRivas

Added one HOS Tri-Patch test to the list.

Change 49796 on 2002/09/05 by frivas@FL_FRivas

Initial checkin of HOS Tri-Patch test. Isolates just two patches (each with 10 control points) to see if there are any seams or inconsistancies on the boundry between patches.
Implements lighting and texturing.

Change 49790 on 2002/09/05 by mkelly@fl_mkelly_r400_win_laptop

1,2,4,6,8 MSAA rectangle list with zbuffering, all vertex ordering and verts falling on
pixel center and non-pixel centers

Change 49781 on 2002/09/05 by mkelly@fl_mkelly_r400_win_laptop

1,2,3,4 MSAA rectangle list, all vertex orders

Change 49752 on 2002/09/05 by mkelly@fl_mkelly_r400_win_laptop

Update comments, add 6 MSAA rectangle list tests

Change 49749 on 2002/09/05 by omesh@omesh_r400_linux_marlboro_release

Added NUM_MULTIWRITES programming (=3) to enable rendering to all 4 Color Buffers from 1 SX Color Export,
using the Color Channel Masks. Verified that the color masking does occur in the emulator in all 4 buffers,
visually, but this also uncovered a bug (Bugzilla Bug #334) with vertex color sequence being out of order in
reference to vertex positions, for Color Buffer 0.

Change 49730 on 2002/09/05 by mkelly@fl_mkelly_r400_win_laptop

8 MSAA rectangle list, all vertex orders

Change 49723 on 2002/09/05 by frivas@FL_FRivas

Added two new HOS tests to the list.

Change 49715 on 2002/09/05 by frivas@FL_FRivas

Initial check in of HOS Line-Patch. This test will be updated soon.

Change 49706 on 2002/09/05 by mkelly@fl_mkelly_r400_win_laptop

SC test, check pixel mask is either 0x00 or 0xFF when MSAA and JSS
are disabled and MSAA_NUM_SAMPLES = 7 at the sc_quad_pair_proc_out.dmp level

Change 49705 on 2002/09/05 by abeaudin@abeaudin_r400_win_marlboro

made gcc hardware accurate - swapped red and blue - made new golden images

Change 49704 on 2002/09/05 by mkelly@fl_mkelly_r400_win_laptop

Log a local sc bug

Change 49685 on 2002/09/05 by frivas@FL_FRivas

Initial checkin of HOS Tri-Patch test. There are 8 patches with 10 control points each. Implements lighting and texturing.

Change 49659 on 2002/09/05 by ygiang@ygiang_r400_win_marlboro_p4

fixed: triangles size for hardware

Change 49618 on 2002/09/05 by kmahler@kmahler_r400_win_devel_views

Fixed processing of VS_PROGRAM_CTRL's VS_EXPORT_MODE in render state.

Also, fixed pixel file name in test.

Change 49595 on 2002/09/04 by csampayo@fl_csampayo_r400

1. Added bug test case
2. Updated bug# 43 on Bug Tracker

Change 49580 on 2002/09/04 by omesh@ma_omesh

Added code to fill and dump all 4 color buffers instead of just the default (color buffer0). I still use the framebuffer class in Primlib as it supports the features I need. Besides the color surface class isn't ready to be used for multiple color buffers yet.

The emulator visual outcome is as yet untested for these multiwrite testcases.

Change 49574 on 2002/09/04 by smoss@smoss_crayola_win

SU test

Change 49532 on 2002/09/04 by kevino@kevino_r400_win_marlboro

Added border size case, but it isn't right yet.
Added exp_adjust_all cases ranging from -32 to 32.

Change 49525 on 2002/09/04 by mkelly@fl_mkelly_r400_win_laptop

Change test name, update tracker...

Change 49517 on 2002/09/04 by mkelly@fl_mkelly_r400_win_laptop

Textured line, 8 MSAA, various orientations

Change 49474 on 2002/09/04 by kevino@kevino_r400_win_marlboro

Added force_bc_w_to_max testcases, and shader program that puts A in R channel so can see results easily.

Change 49452 on 2002/09/04 by mkelly@fl_mkelly_r400_win_laptop

Update second setting of repeat count

Change 49375 on 2002/09/03 by kevino@kevino_r400_win_marlboro

Added testcases that turn off tiling for rectangular and non-rectangular textures

Change 49364 on 2002/09/03 by kevino@kevino_r400_win_marlboro

Added testcases:
more filter cases
2D Aniso
Nearest_Clamp_Policy OGL mode test

Change 49354 on 2002/09/03 by mkelly@fl_mkelly_r400_win_laptop

Initial check of MSAA centermost determination.

Change 49353 on 2002/09/03 by llefebvr@llefebvre_laptop_r400_emu

Added Idle0 and Idle1_7 functions for SQ idle status report.

Change 49352 on 2002/09/03 by omesh@omesh_r400_linux_marlboro_release

Changed background color to black so visual result of color masking is evident. Alicia pointed out that masking is supposed

to occur in the operation of: Destination Color = (Destination Color & ~COLOR_MASK) + (Source Color & COLOR_MASK), not as Destination

Color = Source Color & COLOR_MASK. So, the results of the emulator were correct, hence this bug is invalid.

Change 49272 on 2002/09/03 by frivas@FL_FRivas

Update. There was a typo in the spelling of the data file that the test reads.

Change 49259 on 2002/09/03 by kevino@kevino_r400_sun_marlboro

Fix texture name, add 1024x1024 size

Change 49255 on 2002/09/03 by jhoule@jhoule_r400_win_marlboro

Changed all .compare(..) to .substr(..) == string() to help Linux execution.
Added support for specified miplevel when loading image.
Corrected erroneous error message.

Change 49250 on 2002/09/03 by kevino@kevino_r400_sun_marlboro

1024x1024 texture

Change 49171 on 2002/08/31 by ashishs@fl_ashishs_r400_win

update

Change 49170 on 2002/08/31 by ashishs@fl_ashishs_r400_win

CL test:

The functionality is identical to r400cl_frustum_00 with the exception that it intentionally cycles a postion through all 27 possible clip zones, validating clip operation at each zone.

The clip zone possibilities are:
No Clipping, Right [R], Left[L], Far[F], Near[N], Top[T], Bottom[B],
RT, RB, RF, RFT, RFB, RN, RNT, RNB, LT, LB, LF, LFT, LFB, LN, LNT,
LNB, NT, NB, FT, FB

Change 49164 on 2002/08/31 by ashishs@fl_ashishs_r400_win

CL test:

The functionality is identical to r400cl_frustum_00 with the exception that it intentionally cycles a postion through all 27 possible clip zones, validating clip operation at each zone.

The clip zone possibilities are:
No Clipping, Right [R], Left[L], Far[F], Near[N], Top[T], Bottom[B],
RT, RB, RF, RFT, RFB, RN, RNT, RNB, LT, LB, LF, LFT, LFB, LN, LNT,
LNB, NT, NB, FT, FB

Change 49069 on 2002/08/30 by mkelly@fl_mkelly_r400_win_laptop

Enabling/Disabling control of tp_sq.dmp file for validation

Change 49068 on 2002/08/30 by mkelly@fl_mkelly_r400_win_laptop

Update test to pass HW, but waiting on PrimLib implementation for other tests.

Change 49041 on 2002/08/30 by ygiang@ygiang_r400_test_marlboro

package type changed

Change 49034 on 2002/08/30 by csampayo@fl_csampayo_r400

1. Added 4 new VGT tests for suppress eop function
2. Updated test_list accordingly

Change 49027 on 2002/08/30 by kevino@kevino_r400_win_marlboro

64x64 map

Change 49021 on 2002/08/30 by kevino@kevino_r400_win_marlboro

Added smaller dilbert textures (created with dumpPPM function in uber_map.cpp)

Change 48991 on 2002/08/30 by kevino@kevino_r400_win_marlboro

Made colors different than checker_32x32

Change 48986 on 2002/08/30 by kevino@kevino_r400_win_marlboro

Changed tex size field to 16x16

Change 48985 on 2002/08/30 by kevino@kevino_r400_win_marlboro

16x16 color checker pattern

Change 48981 on 2002/08/30 by kevino@kevino_r400_win_marlboro

Tried to fix checkerboard

Change 48979 on 2002/08/30 by kevino@kevino_r400_win_marlboro

Checkerboard texture

Change 48960 on 2002/08/30 by mkelly@fl_mkelly_r400_win_laptop

Textured line list

Change 48959 on 2002/08/30 by mkelly@fl_mkelly_r400_win_laptop

Match SC spec (repeat count -1)

Change 48958 on 2002/08/30 by mkelly@fl_mkelly_r400_win_laptop

Change stipple repeat count to (count - 1) to match SC spec.

Change 48922 on 2002/08/29 by kevino@kevino_r400_win_marlboro

cleaned up sp filenames, go to only sending 1 tex coord in sp file when needed.

Change 48919 on 2002/08/29 by kevino@kevino_r400_win_marlboro

Added set dest height command.

Change 48905 on 2002/08/29 by kevino@kevino_r400_win_marlboro

Tests for multibuffer assert and seg-fault problem

Change 48889 on 2002/08/29 by omesh@ma_omesh

Tweaked the TRIANGLE_COVERAGE (now = 1.0) parameter in these tests to recreate the old large triangle parallel testcases (to be used to test swamping RB, I suppose).

Change 48876 on 2002/08/29 by kevino@kevino_r400_win_marlboro

Fixed tex clause number for last 2 tex fetches

Change 48865 on 2002/08/29 by mkelly@fl_mkelly_r400_win_laptop

* Basic multi-texture tests
* 16 parameter, 64 dword texture test
* Expand VFD to handle up to 16 textures,
see r400sc_tri_16_par_64_dwords_01.cpp for example useage

Change 48847 on 2002/08/29 by mkelly@fl_mkelly_r400_win_laptop

Update to match SC range of 0 - 255, previously 1 - 256

Change 48818 on 2002/08/29 by kevino@kevino_r400_win_marlboro

fixe num of fb's to be -1 in multiwrite

Change 48813 on 2002/08/29 by kevino@kevino_r400_win_marlboro

added rb_assert test case.

Change 48804 on 2002/08/29 by kevino@kevino_r400_win_marlboro

Updated tri_juice to use enum, not float.

Change 48803 on 2002/08/29 by frivas@FL_FRivas

Added two RECT patch tests to the test list.

Change 48802 on 2002/08/29 by frivas@FL_FRivas

Deleting these two tests because there was a change in their names and descriptions inside the source code. Basically, they've just been renamed.

Change 48798 on 2002/08/29 by frivas@FL_FRivas

Initial check-in of two HOS RECT Patch tests. These tests render the Utah Teapot. One using Bezier interpolation for the patches, and the other using bi-linear interpolation. Both use infinite lighting, texturing, and a tessellation level of 14.9.

Change 48790 on 2002/08/29 by vgoel@fl_vgoel2

added PNT adaptive tessellation test using vertex export

Change 48785 on 2002/08/29 by kevino@kevino_r400_win_marlboro

Added file which is needed by tp_multitexture_01.cpp

Change 48781 on 2002/08/29 by kevino@kevino_r400_win_marlboro

Added 2D clamp mode tests and support textures
parameterize the number of framebuffers
branch 4 color buffer shader pipe program from the original
reset the standard test results to 1 texture, 1 color buffer

Change 48769 on 2002/08/29 by kevino@kevino_r400_sun_marlboro

2x2 went to 104x104 when I saved it after expanding the image, so fixed back to 2x2

Change 48765 on 2002/08/29 by smoss@smoss_crayola_win

Golds with tile enabled

Change 48755 on 2002/08/29 by kevino@kevino_r400_sun_marlboro

Cleaned up smalltex textures

Change 48750 on 2002/08/29 by kevino@kevino_r400_win_marlboro

Small textures for use with clamp mode tests

Change 48735 on 2002/08/29 by ygiang@ygiang_r400_test_marlboro

changed:buffer size

Change 48730 on 2002/08/29 by kevino@kevino_r400_win_marlboro

num_multiwrites should be base 0, so for 4 multiwrites, this should be set to 3.

Change 48693 on 2002/08/28 by omesh@ma_omesh

Added another 30 testcases for Color Channel based masking tests. The tests compile on the emulator, but don't produce the right results on the emulator (nothing renders). Will debug test/emulator later.

Change 48673 on 2002/08/28 by csampayo@fl_csampayo_r400

Updated number of cases

Change 48658 on 2002/08/28 by smoss@smoss_crayola_win

Golds

Change 48645 on 2002/08/28 by kevino@kevino_r400_win_marlboro

Added basic foundations for perl-generation of testcase lists
Added basics for 4 color buffers, but if try to export in the shader program, it asserts.

Change 48639 on 2002/08/28 by vgoel@fl_vgoel2

deleted r400vgt_hos_rpatch_02_pix and _vtx shaders

Change 48638 on 2002/08/28 by vgoel@fl_vgoel2

added few more tests (hos and tone mapping, stereo vision)

Change 48606 on 2002/08/28 by georgev@devel_georgevhw_r400_lin_marlboro

Added more support for random stuff.

Change 48572 on 2002/08/28 by mkelly@fl_mkelly_r400_win_laptop

Update comments

Change 48541 on 2002/08/28 by omesh@ma_omesh

Split another 10 testcases. Compiled and ran them on the emulator, though haven't yet verified visual result (4_4_4_4 format).

Change 48537 on 2002/08/28 by mkelly@fl_mkelly_r400_win_laptop

JSS 4x4 simple triangle

Change 48535 on 2002/08/28 by smoss@smoss_crayola_win

golds

Change 48501 on 2002/08/28 by frivas@FL_FRivas

Initial check-in of a Rect Patch test. It generates a teapot.

Change 48487 on 2002/08/28 by ashishs@fl_ashishs_r400_win

CL tests: adding more UCP combo tests and updating tracker.

Change 48480 on 2002/08/28 by kevino@kevino_r400_win_marlboro

Moved functions to header file

Change 48479 on 2002/08/28 by kevino@kevino_r400_win_marlboro

Added functions to create pix shader programs on the fly so can parameterize TFetchInstr override values
Added 2 simple tesecases for this as well.

Change 48474 on 2002/08/28 by kevino@kevino_r400_win_marlboro

New names for shader pipe programs for tp_multitexture_01

Change 48448 on 2002/08/27 by smoss@smoss_crayola_win

Add test for regress_e

Change 48435 on 2002/08/27 by mkelly@fl_mkelly_r400_win_laptop

Comment out r400sc_point_list_06 until bug fixed

Change 48434 on 2002/08/27 by mkelly@fl_mkelly_r400_win_laptop

Update for new MSAA samples

Change 48430 on 2002/08/27 by mkelly@fl_mkelly_r400_win_laptop

Update to new MSAA sample selects

Change 48402 on 2002/08/27 by csampayo@fl_csampayo_r400

1. Updated test VFD for sourcing tex coord from vertex
2. Updated Bug Tracker accordingly

Change 48393 on 2002/08/27 by mkelly@fl_mkelly_r400_win_laptop

Update for JSS sample 3 - 4 swap

Change 48355 on 2002/08/27 by llefebvr@llefebvre_laptop_r400_emu

For some reason, the test was using the zw coordinates to fetch the texture. They should have used the xy coordinates as the vertex shader only exports those coordinates.

Change 48352 on 2002/08/27 by mkelly@fl_mkelly_r400_win_laptop

JSS Update for sample 3 and 4 swap

Change 48344 on 2002/08/27 by mkelly@fl_mkelly_r400_win_laptop

Update to add test to regress_e r400sc_jss_4x4_fc_02

Change 48339 on 2002/08/27 by mkelly@fl_mkelly_r400_win_laptop

Another quick test needed for SC regress_e

Change 48329 on 2002/08/27 by ygiang@ygiang_r400_win_marlboro_p4

changed:buffer size

Change 48328 on 2002/08/27 by ygiang@ygiang_r400_win_marlboro_p4

changed:buffer size

Change 48327 on 2002/08/27 by ygiang@ygiang_r400_win_marlboro_p4

added: more sp tests

Change 48322 on 2002/08/27 by ashishs@fl_ashishs_r400_win

update

Change 48320 on 2002/08/27 by ashishs@fl_ashishs_r400_win

CL test: adding more UCP combo tests

Change 48304 on 2002/08/27 by fhsien@fhsien_r400_linux_marlboro

Update tests with testcases for smaller framebuffer/images

Change 48291 on 2002/08/27 by csampayo@fl_csampayo_r400

Moving bug case to proper place

Change 48264 on 2002/08/27 by csampayo@fl_csampayo_r400

Bug test case

Change 48259 on 2002/08/27 by csampayo@fl_csampayo_r400

1. Added 1 new SU point sprite test
2. Updated test_list and test tracker accordingly

Change 48245 on 2002/08/27 by mkelly@fl_mkelly_r400_win_laptop

Update .sp name to 02

Change 48229 on 2002/08/27 by fhsien@fhsien_r400_linux_marlboro

Update with small image test cases

Change 48186 on 2002/08/26 by ctaylor@fl_ctaylor_r400_win_marlboro

Change Barycentric logic to correct 3-input adder carry-in problem.
Added rounding to BarycBack multiplier for more accuracy on directed tests.
Updated golds for 2 SC, 2 VGT and 2 SU tests which were affected by above changes.
Added dumps for quad covered and cleaned up some dump file headers.
Fixed zmin/zmax poly offset code for sign extend bug.
Changed 6 and 8 Sample AA sample locations for new 6-sample grid.
Added 2 and 3 sample AA sample locations.
Fixed .cpp of two regression SC tests affected by AA sample location changes.

Change 48173 on 2002/08/26 by vgoel@fl_vgoel2

added data file for tri-patches

Change 48171 on 2002/08/26 by vgoel@fl_vgoel2

added tri-patch displaying an object

Change 48107 on 2002/08/26 by mkelly@fl_mkelly_r400_win_laptop

tile control registry files

Change 48104 on 2002/08/26 by mkelly@fl_mkelly_r400_win_laptop

Re-included SC stipple test

Change 48098 on 2002/08/26 by omesh@omesh_r400_linux_marlboro_release

Changed COVERAGE settings to render smaller triangles in scalable tests, to reduce simulation time
and storage space requirements for results. Made most of the triangles about 1/16th their size.

Change 48091 on 2002/08/26 by jhoule@jhoule_r400_win_marlboro

New TRI_JUICE specification.

Change 48080 on 2002/08/26 by llefebvr@llefebvre_laptop_r400_emu

There was an indexing problem with the writting of the STs in the GPRs.

Change 48071 on 2002/08/26 by kevino@kevino_r400_win_marlboro

Cleaned up, added format_2101010.

Change 48054 on 2002/08/26 by jhoule@jhoule_r400_win_marlboro

Setting both MIN/MAG filters

Change 48046 on 2002/08/26 by ashishs@fl_ashishs_r400_win

update to testlist

Change 48039 on 2002/08/26 by mkelly@fl_mkelly_r400_win_laptop

Memory crash when shader file doesn't exist..

Change 48026 on 2002/08/26 by kevino@kevino_r400_win_marlboro

added format_5_6_5 case

Change 48022 on 2002/08/26 by mkelly@fl_mkelly_r400_win_laptop

Archiving potential bug in SQ ST determination.

Change 48021 on 2002/08/26 by mkelly@fl_mkelly_r400_win_laptop

Textured line where ST are exported directly to RGBA to visulize texture coordinates in color. Current there is a bug in the LLC quad pixel ST value when the primitive is not quad-aligned.

Change 47986 on 2002/08/26 by kevino@kevino_r400_win_marlboro

Added load_vertex function to make it easier.

Change 47980 on 2002/08/26 by kevino@kevino_r400_win_marlboro

cleaned up baee and mip offset code

Change 47967 on 2002/08/25 by csampayo@fl_csampayo_r400

1. Added 1 new SU point sprite tests
2. Updated test_list and the test tracker accordingly

Change 47946 on 2002/08/24 by tho@tho_r400_win

Changed MEM_BASE and BIOS_BASE setting in bif_init.cpp in chip library.
Take out one failing test (not sure how to fix the test) from regression.
Updated golden images - all the mem_framebuf_r are modified by a script, all the rd_r files are done manully.
Hope this submit won't break in the middle.

Change 47918 on 2002/08/23 by ashishs@fl_ashishs_r400_win

pointlist test, with 8 textures and 1 color. texture currently disabled through VFD, will be incorporated soon.

Change 47900 on 2002/08/23 by mkelly@fl_mkelly_r400_win_laptop

Update test so it will compile, bug still exists...

Change 47899 on 2002/08/23 by smoss@smoss_crayola_linux_orl_regress

update

Change 47895 on 2002/08/23 by mkelly@fl_mkelly_r400_win_laptop

* Line list, start vertex at -8190 in HW space, clipped at -4096 Screen Space
* Line list, textured

Change 47891 on 2002/08/23 by kevino@kevino_r400_win_marlboro

generalize tp_multitexture_01 to parameterize all of the TFetchConst fields

Change 47789 on 2002/08/23 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 47788 on 2002/08/23 by mkelly@fl_mkelly_r400_win_laptop

Bug caused by primlib/render_state/render_register_state.cpp#42

Change 47756 on 2002/08/23 by mkelly@fl_mkelly_r400_win_laptop

PrimLib bug

Change 47731 on 2002/08/23 by mdoggett@mdoggett_r400_linux_local

Updated dxtc test to work with changed texture constant. Also changed image from window to boards as boards shows the compression artifacts of dxtc1.

Change 47729 on 2002/08/23 by vgoel@fl_vgoel2

added simple line patch test

Change 47728 on 2002/08/23 by vgoel@fl_vgoel2

added complex teapot test for rect patches

Change 47717 on 2002/08/23 by mkelly@fl_mkelly_r400_win_laptop

Simple single line list

Change 47623 on 2002/08/22 by llefebvr@llefebvre_laptop_r400_emu

using resize instead of push_back

Change 47622 on 2002/08/22 by kevino@kevino_r400_win_marlboro

Added a couple more testcases to tp_aniso
Added default aniso field to tp_unsigned32_01 and tp_multitexture_0

Change 47607 on 2002/08/22 by kevino@kevino_r400_win_marlboro

1st cut at aniso test. Not sure if it is working in emu yet.

Change 47575 on 2002/08/22 by georgev@devel_georgevhw_r400_lin_marlboro

Fixed frame buffer size.

Change 47534 on 2002/08/22 by georgev@devel_georgevhw_r400_lin_marlboro

More changes.

Change 47531 on 2002/08/22 by kevino@kevino_r400_sun_marlboro

Texture that shows text (nice for aniso)

Change 47522 on 2002/08/22 by csampayo@fl_csampayo_r400

1. Added 10 new VGT multi-context tests
2. Updated test_list and the test tracker accordingly

Change 47507 on 2002/08/22 by frivas@FL_FRivas

Change to anonymous struct definition in "Matrix_Class" to allow compiling under Linux.

Change 47506 on 2002/08/22 by kevino@kevino_r400_win_marlboro

added f to float values so don't get double to float warnings

Change 47493 on 2002/08/22 by mkelly@fl_mkelly_r400_win_laptop

Finalized second poly offset test

Change 47491 on 2002/08/22 by mkelly@fl_mkelly_r400_win_laptop

Add two test descriptions, update comments in test

Change 47454 on 2002/08/21 by ashishs@fl_ashishs_r400_win

CL test:r400cl_ucp_pointlist_01
64 point sprites with 6 textures and 6 ucp's enabled with point size set in PA_SU state register and also in PA_CL_POINT registers for clipping. Point Sprite UCP mode set to '3' viz. always expand and clip.

Change 47443 on 2002/08/21 by mkelly@fl_mkelly_r400_win_laptop

One more time..

Change 47438 on 2002/08/21 by mkelly@fl_mkelly_r400_win_laptop

Try again...

Change 47437 on 2002/08/21 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 47431 on 2002/08/21 by mkelly@fl_mkelly_r400_win_laptop

del

Change 47429 on 2002/08/21 by mkelly@fl_mkelly_r400_win_laptop

golds...

Change 47423 on 2002/08/21 by mkelly@fl_mkelly_r400_win_laptop

Add to SC regress_e

Change 47414 on 2002/08/21 by mkelly@fl_mkelly_r400_win_laptop

Fix a typo, changed bool to uint32 on MAX_DISTANCE for aa

Change 47406 on 2002/08/21 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 47404 on 2002/08/21 by mkelly@fl_mkelly_r400_win_laptop

Increasing SC regress_e with some very quick tests to give more coverage,

Change 47380 on 2002/08/21 by kevino@kevino_r400_linux_marlboro

Added a random texture case where the vertices are left alone

Change 47378 on 2002/08/21 by mkelly@fl_mkelly_r400_win_laptop

DEPTH_16 and DISP_Y_DIM = 64

Change 47377 on 2002/08/21 by mkelly@fl_mkelly_r400_win_laptop

Update to include R400HardwareAccurate = 0

Change 47376 on 2002/08/21 by mkelly@fl_mkelly_r400_win_laptop

Update tests DISP_Y_DIM to 64 to work around Primlib issue. Does not
affect functional part of test but permits HW simulation to continue

Change 47369 on 2002/08/21 by kevino@kevino_r400_win_marlboro

Added several basic 1-4 text testcases and a shader program which doesn't blend in the color

Change 47362 on 2002/08/21 by llefebvr@llefebvre_laptop_r400_emu

Dumplib now supports : FMT_8, FMT_8_8_8_8, FMT_5_6_5, FMT_4_4_4_4, FMT_2_10_10_10, FMT_1_5_5_5, FMT_16_16, FMT_32_32_32_32_FLOAT, FMT_16_16_16_16, DEPTH_16, DEPTH_24_8, DEPTH_24_8_FLOAT.

Change 47340 on 2002/08/21 by kevino@kevino_r400_win_marlboro

1st simple testcase files for tp_multitexture_01

Change 47338 on 2002/08/21 by kevino@kevino_r400_win_marlboro

Created XYZW_RGBA_STQ4 data structure w/ 4 STQ coords
Set up multitexture test to draw 1-4 textures. (Need more pix.sp files for more than that)

Change 47333 on 2002/08/21 by frivas@FL_FRivas

update

Change 47330 on 2002/08/21 by frivas@FL_FRivas

Change to annonymous struct definition in "Matrix_Class" to allow compiling in Linux.

Change 47328 on 2002/08/21 by abeaudin@abeaudin_r400_win_marlboro

fixed ptich problem.

Change 47325 on 2002/08/21 by frivas@FL_FRivas

Change to annonymous struct definition in "Matrix_Class" to allow compiling on Linux.

Change 47318 on 2002/08/21 by frivas@FL_FRivas

Change to annonymous struct definition in "Matrix_Class" to allow compling under Linux.

Change 47308 on 2002/08/21 by frivas@FL_FRivas

Change to annonymous struct in "Matrix_Class" to allow compiling under Linux.

Change 47302 on 2002/08/21 by frivas@FL_FRivas

Change to annonymous struct defined in "Matrix_Class" to allow compiling under Linux.

Change 47301 on 2002/08/21 by vgoel@fl_vgoel2

added first tri-patch test

Change 47281 on 2002/08/21 by frivas@FL_FRivas

Changed annonymous struct definition in "Matrix_Class" to allow compling under Linux.

Change 47269 on 2002/08/21 by frivas@FL_FRivas

Change to annonymous struct definition in "Matrix_Class" to allow compilation under Linux.

Change 47177 on 2002/08/20 by csampayo@fl_csampayo2_r400

Updates for surface height constraint

Change 47167 on 2002/08/20 by mkelly@fl_mkelly_r400_win_laptop

Update test to run several 12 seg concentric circles...

Change 47163 on 2002/08/20 by mkelly@fl_mkelly_r400_win_laptop

Count double hit pixels if more than one occurs in a quad.

Change 47146 on 2002/08/20 by kevino@kevino_r400_win_marlboro

cast (*(tfc[texnum]) to TFETCH_CONSTANT_REGISTER before sending it to gRenderState->Set()

Change 47130 on 2002/08/20 by llefebvr@llefebvre_laptop_r400_emu

Now returning vectors of strings

Change 47128 on 2002/08/20 by mkelly@fl_mkelly_r400_win_laptop

Update comments in test...

Change 47127 on 2002/08/20 by mkelly@fl_mkelly_r400_win_laptop

Line list concentric circle, line width increased to see double hit pixels

Change 47117 on 2002/08/20 by kevino@kevino_r400_win_marlboro

Cleaned up, added more debug info. Works on Linux, but not windows.

Change 47106 on 2002/08/20 by kevino@kevino_r400_win_marlboro

Multitexture test. Rev1 compiles, but has a pointer error with gRenderState->Set(*tfc[texnum]);

Change 47098 on 2002/08/20 by mkelly@fl_mkelly_r400_win_laptop

Added object for keeping track of multi-processed pixels in the SC,
modified test to display multi-hit pixels

Change 47024 on 2002/08/20 by kevino@kevino_r400_win_marlboro

Add basemap to randomize_FILTER function. Also set parameters for miptint functions.

Change 47022 on 2002/08/20 by kevino@kevino_r400_win_marlboro

Added Mip_BaseMap to several testcases

Change 46991 on 2002/08/20 by kevino@kevino_r400_win_marlboro

Added point and linear mipmap modes

Change 46914 on 2002/08/19 by csampayo@fl_csampayo_r400

1. Added 2 new VGT test for multi context coverage
2. Updated test_list and the test tracker appropriately

Change 46895 on 2002/08/19 by mkelly@fl_mkelly_r400_win_laptop

Primlib utility to check for double hit pixels in sc_quad_pair_proc_out.dmp

Concentric circle test to do a preliminary check on the new utility

Change 46866 on 2002/08/19 by kevino@kevino_r400_win_marlboro

New testcases for tp_unsigned32_01.cpp

Change 46857 on 2002/08/19 by kevino@kevino_r400_win_marlboro

Added non-rectangular testcases
Cleaned up some oduble->float warnings by specifying floats with 1.0f instead of 1.0
Added tint funtion to testcases
expanded texture_manager tintarray to be 15x4 to cover up to 16K 1D textures

Change 46823 on 2002/08/19 by smoss@smoss_crayola_win

removed duplicate case

Change 46777 on 2002/08/19 by csampayo@fl_csampayo_r400

1. Added 4 new SU polymode degenerate tests
2. Updated test_list and test tracker accordingly

Change 46753 on 2002/08/19 by kevino@kevino_r400_linux_marlboro

Added EstesPark images (mostly non-square)

Change 46729 on 2002/08/19 by kevino@kevino_r400_win_marlboro

define prim size to make changing it easier

Change 46695 on 2002/08/18 by csampayo@fl_csampayo_r400

1. Added 8 new SU polymode tests
2. Updated test_list and the test tracker accordingly

Change 46633 on 2002/08/16 by kevino@kevino_r400_win_marlboro

Fixed some maxmip values, put in commented out tintMipChain call.

Change 46598 on 2002/08/16 by kevino@kevino_r400_win_marlboro

Added some non-power2 texture cases with various filtering

Change 46586 on 2002/08/16 by kevino@kevino_r400_linux_marlboro

Non-power2 sized textures

Change 46572 on 2002/08/16 by omesh@omesh_r400_linux_marlboro_only_devel

Changed format of render and destination to 4444 instead of 8888, so
that visual effects of the dither tests can be seen. Also switched to

using the new Surface format class of primlib that supports fill and
dump of the 4444 format. Still haven't seen the visual effects of dither
(Dumpview comes up too slowly on a fully loaded lmcs* or lmct* machine
today!!)

Change 46518 on 2002/08/16 by georgev@devel_georgevhw_r400_lin_marlboro

Change to new format.

Change 46515 on 2002/08/16 by georgev@devel_georgevhw_r400_lin_marlboro

Put in missing semi-colon.

Change 46505 on 2002/08/16 by jhoule@jhoule_r400_win_marlboro

Added support for {MIN|MAX}_MIP_LEVEL and LOD_BIAS_{H|V}.
Added generateMipmapChain() function.

Change 46391 on 2002/08/15 by mkelly@fl_mkelly_r400_win_laptop

Possible interpolator bug...

Change 46353 on 2002/08/15 by kevino@kevino_r400_linux_marlboro_release

Changed maxmiplevel to 9 since the texture is 512x512

Change 46349 on 2002/08/15 by vgoel@fl_vgoel2

changed the tessellation level to one

Change 46347 on 2002/08/15 by kevino@kevino_r400_win_marlboro

Set the maxmiplevel to 10

Change 46257 on 2002/08/15 by mkelly@fl_mkelly_r400_win_laptop

Modified SC regress_e tests to reduce emulate time by 66% (from 1:41 to 0:47 total for five tests). In general, less cases are tested but specific
functional / operational check still occurs.

Change 46255 on 2002/08/15 by csampayo@fl_csampayo_r400

1. Added 3 new SU tests
2. Updated test_list and the test tracker accordingly

Change 46245 on 2002/08/15 by abeaudin@abeaudin_r400_win_marlboro

fixed dither bug - rounding was not being done

Change 46186 on 2002/08/14 by georgev@devel_georgevhw_r400_lin_marlboro

Changed predicate tests. Began serialize test.

Change 46124 on 2002/08/14 by omesh@omesh_r400_linux_marlboro_release

Fixed some bugs related to placement of geometry on screen and also add
some more random testcases. I tried running the testcases on linux and
they seem to execute, although the emulator doesn't produce the right
result.

Change 46120 on 2002/08/14 by kevino@kevino_r400_linux_marlboro

dilbert texture in ppm ascii format to make sure windows raw fix doesn't break ascii mode

Change 46109 on 2002/08/14 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 46106 on 2002/08/14 by mkelly@fl_mkelly_r400_win_laptop

More bres control tests...

Change 46015 on 2002/08/14 by omesh@ma_omesh

Added special testcases for LUT based color dither, for more extensive testing. Compiled and run on Windows. Not yet checked on Linux.

Change 45945 on 2002/08/14 by jhoule@jhoule_r400_win_marlboro

Converted dilbert_128x128.ppm into ascii PPM format.

Change 45899 on 2002/08/14 by mkelly@fl_mkelly_r400_win_laptop

Line Strip stipple packet switching in progress...

Change 45816 on 2002/08/13 by csampayo@fl_csampayo_r400

Update to properly set X and Y radius for per vertex point size

Change 45815 on 2002/08/13 by mkelly@fl_mkelly_r400_win_laptop

50% done...

Change 45788 on 2002/08/13 by omesh@ma_omesh

Added Alpha Dither testcases (5). They compile and run on Windows. Haven't checked them on Linux yet.

Change 45761 on 2002/08/13 by omesh@omesh_r400_linux_marlboro_release

Fixed a bug with calculating colors to be rendered to bring out dither
effects. Ran the test on Linux and verified that the test is doing what
it should be, even though the emulator doesn't produce the right result.

Change 45748 on 2002/08/13 by omesh@ma_omesh

Added extensive testcases for TRUNC and ROUND cases of color dither. There are 5 testcases (including 3 random). I have run them on Windows and have yet to run them on Linux.

Change 45736 on 2002/08/13 by csampayo@fl_csampayo2_r400

Updates for latest primlib surface change

Change 45684 on 2002/08/13 by csampayo@fl_csampayo_r400

Add bug case

Change 45673 on 2002/08/13 by mkelly@fl_mkelly_r400_win_laptop

Expand line width basic functionality...

Change 45643 on 2002/08/13 by frivas@FL_FRivas

Update to output image size (32-bit alligned) and Z-Buffer set up (new R400 DepthFormat enums).

Change 45633 on 2002/08/13 by ashishs@fl_ashishs_r400_win

commented out a test for further investigation

Change 45631 on 2002/08/13 by ashishs@fl_ashishs_r400_win

changed to latest register spec

Change 45591 on 2002/08/12 by ygiang@ygiang_r400_win_marlboro_p4

modified for ken mahler to debug

Change 45580 on 2002/08/12 by omesh@omesh_r400_linux_marlboro_release

Fixed a compiler error on linux. The test, however, still doesn't show
dither to be working.

Change 45577 on 2002/08/12 by omesh@ma_omesh

Made a basic dither test that tests all 3 modes for Color/Alpha dither. The test compiles on Windows, but I haven't yet checked if it is running on Linux.

Change 45496 on 2002/08/12 by smoss@smoss_crayola_win

Added vgt dma dump files

Change 45495 on 2002/08/12 by omesh@ma_omesh

Fixed a typo that was keeping the test from compiling.

Change 45486 on 2002/08/12 by ygiang@ygiang_r400_win_marlboro_p4

added: more SP opcode tests

Change 45485 on 2002/08/12 by jhoule@jhoule_r400_win_marlboro

Resubmitted golden image, changing type to binary (Windows regression was broken otherwise).

Change 45456 on 2002/08/12 by kryan@kryan_r400_win_marlboro

Updating all golden images in the R400 Emulator regression suite

except those under the test_lib/src/chip/quickemu/ directory.

Modifying chip/perf/ tests:

- Update golden images to include new R400 header information,

  and since ColorSurface0 is now tiled by default.

primlib_template_simple_triangle.cpp

  - Modified to force it to always use tiled Fill_Solid(), tiled ColorSurface,

    and tell the Dump() function that the ColorSurface is tiled.

primlib_tex_tri.cpp

  - Modified to use new SurfaceFormat FMT_8_8_8_8 for ColorSurface instead

    of old plgx::pixelType.

  - Added new PIXEL_SURFACE object that represents the color surface.

    - Initialize this object with the Fill_Solid() command.

    - Call member function Dump() to dump the memory occupied by

      by this object. This simplifies the Fill/Dump calls.

  - Modified call to Load_Texture_Buffer_And_Write_To_Memory() function

    to disable downsampled mipmap generation, and force fills to be

    tiled when the texture is written to memory.

  - Modified the test to set the appropriate field in the Texture Fetch

    Constant register telling whether the texture is tiled in memory

    or not.

gfx/pa/CL tests

- Update golden images to include new R400 header information,

  and since ColorSurface0 is now tiled by default.

r400cl_clip_space_dx_ogl_01.cpp

  - modified code to calculate visible portion of framebuffer to use

    DISP_PITCH instead of DISP_X_DIM since DISP_PITCH > DISP_X_DIM.

    This will correctly calculate the actual dimensions of the color

    surface that will be rendered to.

  - Updated Dump() call to use DISP_PITCH as width of memory to be

    dumped so that the whole color surface will be dumped into the

    dumpfile. This is necessary so that the viewer can properly

    detile the surface.

gfx/pa/SU tests

- Update golden images to include new R400 header information,

  and since ColorSurface0 is now tiled by default.

gfx/pa/VTE tests

- Update golden images to include new R400 header information,

  and since ColorSurface0 is now tiled by default.

gfx/SC tests

- Update golden images to include new R400 header information,

  and since ColorSurface0 is now tiled by default.

chip/gfx/VGT tests

- Update golden images to include new R400 header information,

  and since ColorSurface0 is now tiled by default.

r400vgt_hos_PNT_cp_qn_disc_14_04_lit_tex_proj_01.cpp

  - Modified to force fills to be tiled.

  - Set field in texture constant register to indicate

    the texture is tiled in memory.

chip/sys/CP tests

- Update golden images to include new R400 header information.

cp_simple_triangle.cpp

- Modified test to tell Dump() that ColorSurface0 is not tiled.

Change 45432 on 2002/08/12 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 45408 on 2002/08/12 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 45399 on 2002/08/12 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 45365 on 2002/08/11 by ygiang@ygiang_r400_win_marlboro_p4

added: new shader tests with some of new opcode

Change 45364 on 2002/08/11 by ygiang@ygiang_r400_win_marlboro_p4

added: dot3 and dot4 shader tests

Change 45362 on 2002/08/11 by omesh@ma_omesh

Split some more testcases.

Change 45360 on 2002/08/11 by omesh@omesh_r400_linux_marlboro_release

Fixed some bugs related to constant color/alpha blending programming.
Also re-verified all results visually.

Change 45326 on 2002/08/09 by kryan@kryan_r400_win_marlboro

Orlando texture tests not in regression suite

r400vte_xy_fmt_01.cpp

- Changed dimensions of dump file so that it will detile properly

when color buffer is tiled.

Change 45316 on 2002/08/09 by csampayo@fl_csampayo_r400

1. Updated image size for tests: r400su_point_sprite_01 and _02
2. Added 4 more SU point sprite tests
3. Updated test_list and test tracker accordingly

Change 45246 on 2002/08/09 by georgev@devel_georgevhw_r400_lin_marlboro

More tests.

Change 45245 on 2002/08/09 by mkelly@fl_mkelly_r400_win_laptop

Enhance comments in test, initialize all clip rects to zero

Change 45242 on 2002/08/09 by ygiang@ygiang_r400_win_marlboro_p4

more for debug

Change 45239 on 2002/08/09 by ygiang@ygiang_r400_win_marlboro_p4

new for debug

Change 45236 on 2002/08/09 by ygiang@ygiang_r400_win_marlboro_p4

added: test for debug

Change 45233 on 2002/08/09 by smoss@smoss_crayola_win

Updated for new code

Change 45230 on 2002/08/09 by mkelly@fl_mkelly_r400_win_laptop

Poly offset 100 packets, varying offset from +0.5 to -0.5

Change 45220 on 2002/08/09 by mkelly@fl_mkelly_r400_win_laptop

Nan retain/kill on 2 vector exports from shader bug...

Change 45169 on 2002/08/09 by frivas@FL_FRivas

Initial check in of input data image for "r400vgt_hos_PNT_edge_detection_01" test.

Change 45168 on 2002/08/09 by frivas@FL_FRivas

Deleted file from Perforce because there is a new input file with a different name.

Change 45167 on 2002/08/09 by frivas@FL_FRivas

Update. Changed name of input data file to
"r400vgt_hos_PNT_edge_detection_001.bmp"

Change 45164 on 2002/08/09 by llefebvr@llefebvre_laptop_r400_emu

Corrected one SP test.

Change 45161 on 2002/08/09 by ygiang@ygiang_r400_win_marlboro_p4

added: vertex pass through with alpha fetch..

Change 45160 on 2002/08/09 by ygiang@ygiang_r400_win_marlboro_p4

added: tests w/ bugs to fix

Change 45142 on 2002/08/09 by jhoule@jhoule_r400_win_marlboro

New test program where things are specified through a test file.

Supports:
- vertex array
- index array (tri and quad lists)
- vertex and pixel shader file specification
- texture (ppm filename, most constants)

Added default files, plus format manipulation test file.

Change 45045 on 2002/08/08 by ashishs@fl_ashishs_r400_win

CL test:
A test to determine if the clip guard band works properly and that trivial reject works.
update to test tracker and test list.

Change 45040 on 2002/08/08 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 45038 on 2002/08/08 by frivas@FL_FRivas

Update. Added edge detector test to test list.

Change 45010 on 2002/08/08 by frivas@FL_FRivas

Initial check in of Canny Edge detector using the pixel shader to implement the algorithm. Also fed through the HOS engine at level 0 discrete tessellation.

Change 44986 on 2002/08/08 by frivas@FL_FRivas

Update. Took out 1 HOS test and added 4 others.

Change 44982 on 2002/08/08 by frivas@FL_FRivas

Initial check in of HOS precision tests.

Change 44980 on 2002/08/08 by frivas@FL_FRivas

Deleted because there is another test that does the same thing with a slightly different name.

Change 44954 on 2002/08/08 by omesh@omesh_r400_linux_marlboro_release

Added a comment about the expected outcome of a working test (Green Triangle) or a possible not working test (Yellow triangle)

Change 44952 on 2002/08/08 by omesh@omesh_r400_linux_marlboro_release

Made a basic fog test that uses a constant fog factor (1.0: Max) in the SP (Pixel Shader) and exports it for use to the RB.
The test doesn't work right now because the emulator doesn't support the bit backing associated with storing/exporting fog factor. Laurent will inform me when he implements the bit packing in the emulator, so that Alicia can test the RB Emulator functioning of fog blending.

Change 44939 on 2002/08/08 by kevino@kevino_r400_linux_marlboro_release

Just another texture map...

Change 44907 on 2002/08/08 by csampayo@fl_csampayo_r400

1. Initial check in of Bug Tracker
2. Added bug test case

Change 44906 on 2002/08/08 by kevino@kevino_r400_linux_marlboro_release

Construct: put dependency on ferret)ctrl.v
SQ_TP_interface.v: emu send single request over 4 cycles (with different pix mask and fetch_addr, etc) , not 1 cycle/command
tp_simple_02: changed to 7x7 primitive instead of 100x100

Change 44896 on 2002/08/08 by frivas@FL_FRivas

Update to tessellation level.

Change 44801 on 2002/08/07 by georgev@devel_georgevhw_r400_lin_marlboro

Calls aren't working, so I signed it in for debug.

Change 44797 on 2002/08/07 by frivas@FL_FRivas

Update to tessellation level (changed to 1.0).

Change 44773 on 2002/08/07 by abeaudin@abeaudin_r400_win_marlboro

fog implementation complete

Change 44761 on 2002/08/07 by omesh@omesh_r400_windowsXp_marlboro_1_

Split some testcases and created new tests. For some reason they don't work on Windows (crash), but they seem to work on Linux, so I'll test these on Linux exclusively.
Fixed some typos in r400rb_alpha_source as well as some bugs.

Change 44734 on 2002/08/07 by kevino@kevino_r400_linux_marlboro_release

Combined tp_simple, tp_2d_clamp, tp_2d_lod_bias, tp_1d_ etc to 1 test with testcases
The trick is to break up the testcases (DEFINE_TEST_CASE) into include files with corresponding testcasename files (DEFINE_TEST_CASE_NAME) so they are manageable.

Change 44731 on 2002/08/07 by kevino@kevino_r400_linux_marlboro_release

changed texture_filename to char* so it would work properly

Change 44726 on 2002/08/07 by mkelly@fl_mkelly_r400_win_laptop

Update, this test currently hangs...

Change 44699 on 2002/08/07 by mdoggett@mdoggett_r400_linux_local

Updated for new texture fetch constant pitch which is now indicates the number of 32 elements.
Also added a makefile for TC tests.

Change 44675 on 2002/08/07 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 44636 on 2002/08/07 by mkelly@fl_mkelly_r400_win_laptop

More JSS coverage...

Change 44616 on 2002/08/07 by kevino@kevino_r400_linux_marlboro_release

1st cut at random

Change 44586 on 2002/08/06 by csampayo@fl_csampayo2_r400

Updated for SQ param gen new feature and associated VFD support

Change 44553 on 2002/08/06 by frivas@FL_FRivas

Initial check-in of HOS PNT tests that use a complex model (human-like head) and tessellates 190 triangles to a level of 14.0 (in discrete case) and 14.99 (in continuous case). Implements perspective projeciton, lighting, and texturing.

Change 44514 on 2002/08/06 by kevino@kevino_r400_linux_marlboro_release

Includes some more include directories (taken from the sq directory)
also includes the -v options, which I believe dump some more info .
This will not work on Windows.

Change 44512 on 2002/08/06 by mkelly@fl_mkelly_r400_win_laptop

Textured pixel shader anti-aliased line example with VFD support.

Change 44501 on 2002/08/06 by ashishs@fl_ashishs_r400_win

changed to latest includes

Change 44495 on 2002/08/06 by ashishs@fl_ashishs_r400_win

corrected error (reverted back to the previous version, accidently changed while testing)

Change 44479 on 2002/08/06 by ashishs@fl_ashishs_r400_win

changed for latest "includes"

Change 44475 on 2002/08/06 by ashishs@fl_ashishs_r400_win

changed to latest "includes"

Change 44458 on 2002/08/06 by ashishs@fl_ashishs_r400_win

update

Change 44450 on 2002/08/06 by ashishs@fl_ashishs_r400_win

CL test: to test clip disable feature

Change 44421 on 2002/08/06 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 44377 on 2002/08/06 by kevino@kevino_r400_linux_marlboro_release

1st LOD test

Change 44350 on 2002/08/06 by llefebvr@llefebvre_laptop_r400_emu

There was a bug in the SQ that made the last iteration of a loop read invalid loop indexes. This is now fixed.

Change 44318 on 2002/08/05 by mkelly@fl_mkelly_r400_win_laptop

Update to work correctly with MOD3 operation...

Change 44303 on 2002/08/05 by omesh@omesh_r400_linux_marlboro_release

Removed comments from lines of code that maintain the destination alpha.
I will file a Bugzilla bug for the emulator for this. Already spoken to
Alicia about the bug.

Change 44292 on 2002/08/05 by fhsien@fhsien_r400_linux_marlboro

ADD Stencil_ZFAIL test

Change 44273 on 2002/08/05 by georgev@ma_georgev

Changed loop variables for windoze compatability.

Change 44265 on 2002/08/05 by kevino@kevino_r400_linux_marlboro_release

modified dilbert.ppm that uses forced colors in image to identify mip level (ie.e. image size level)

Change 44246 on 2002/08/05 by mkelly@fl_mkelly_r400_win_laptop

Update to work in PrimLib...

Change 44167 on 2002/08/05 by mkelly@fl_mkelly_r400_win_laptop

JSS 3x4 unique sample sel testing...

Change 44165 on 2002/08/05 by fhsien@fhsien_r400_linux_marlboro

ADD Stencil_ZPASS test.

Change 44096 on 2002/08/02 by mkelly@fl_mkelly_r400_win_laptop

* Implement vertex kill in the VFD for vertex buffer and constant
* Create a new class called PRIMITIVE_AA for AA dump file analysis
* Update tracker / test list

Change 44063 on 2002/08/02 by mkelly@fl_mkelly_r400_win_laptop

Simple Poly offset check, needs SC fix to work...

Change 44044 on 2002/08/02 by georgev@georgev_r400_linux_marlboro

Added loop tests and modifications to roll and deal to go faster and eliminate a bug.

Change 43966 on 2002/08/02 by mkelly@fl_mkelly_r400_win_laptop

Run line and point prims through JSS and MSAA...

Change 43934 on 2002/08/02 by kevino@kevino_r400_linux_marlboro

Garbage file- never meant to be checked in

Change 43932 on 2002/08/02 by kevino@kevino_r400_linux_marlboro

New textures (linear step in color in X,Y, but with DC offset so can see them)
New 1D test (first cut)

Change 43912 on 2002/08/02 by ashishs@fl_ashishs_r400_win

updated the pointer to shader files

Change 43845 on 2002/08/01 by frivas@FL_FRivas

Update to HOS precision tests to clamp the mipmap level of detail at 1.

Change 43776 on 2002/08/01 by mkelly@fl_mkelly_r400_win_laptop

Bres Cntl test....

Change 43775 on 2002/08/01 by mkelly@fl_mkelly_r400_win_laptop

Bresenham Control, converted R200 test...

Change 43766 on 2002/08/01 by kevino@kevino_r400_linux_marlboro

1st (untedted) cut at a 1D texture test

Change 43754 on 2002/08/01 by frivas@FL_FRivas

Update. Added two HOS tests for a complex model.

Change 43716 on 2002/08/01 by frivas@FL_FRivas

Added HOS auto index quad list test.

Change 43712 on 2002/08/01 by frivas@FL_FRivas

Initial check-in of HOS auto indexed quad list. Uses continuous tessellation at level 14.99 on 16 different objects. Each object is composed of approximately 5 primitives.

Change 43679 on 2002/08/01 by ygiang@ygiang_r400_win_marlboro_p4

added: Simple vertex shader tests

Change 43650 on 2002/08/01 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 43646 on 2002/08/01 by mkelly@fl_mkelly_r400_win_laptop

New...

Change 43645 on 2002/08/01 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 43638 on 2002/08/01 by mkelly@fl_mkelly_r400_win_laptop

JSS 3x4 simple triangle

Change 43613 on 2002/07/31 by abeaudin@abeaudin_r400_win_marlboro

added chroma keying feature

Change 43560 on 2002/07/31 by abeaudin@abeaudin_r400_win_marlboro

new chroma_test

Change 43550 on 2002/07/31 by fhsien@fhsien_r400_linux_marlboro

Update for more tests

Change 43546 on 2002/07/31 by csampayo@fl_csampayo2_r400

1. Added VGT test for provoking vertex and edgeflgas validation
2. Updated test_list

Change 43540 on 2002/07/31 by kevino@kevino_r400_sun_marlboro

Added texture maps of various sizes. Each point has a unique value
The X coord of the texture is in G[3:0], B[7:0].
The Y coord is in R[7:0], G[7:4]

Change 43498 on 2002/07/31 by kevino@kevino_r400_win_marlboro

Branched off clamping cases in tp_simple_02 to tp_2D_clamp_01.
Every combination of S and T clamping is done in a test case. (Add random XY for each of these later)

Change 43493 on 2002/07/31 by mkelly@fl_mkelly_r400_win_laptop

Testing lots of SC features with stipple enabled...

Change 43486 on 2002/07/31 by mkelly@fl_mkelly_r400_win_laptop

For record keeping...

Change 43475 on 2002/07/31 by ygiang@ygiang_r400_win_marlboro_p4

added: last opcode for pixel shader tests

Change 43444 on 2002/07/31 by kevino@kevino_r400_linux_marlboro

Go back to the older test_reg include for now. (New one is commented out)

Change 43438 on 2002/07/31 by kevino@kevino_r400_win_marlboro

Added several additional clamp cases, and some cases with S and T clamps are different.
Loading of different texture sizes still is not fixed.

Change 43433 on 2002/07/31 by ashishs@fl_ashishs_r400_win

update for latest register specs

Change 43422 on 2002/07/31 by kevino@kevino_r400_win_marlboro

Temporary fix- force tex to be dilbert_128x128 since it is having trouble converting the string to char* in linux.
Cleaned up the exta filter and clamping emumerated typed, and use the ones directly from tconst.h

Change 43418 on 2002/07/31 by efong@efong_crayola_linux_cvd

changed from chip/test_regs.h to ar_test/test_regs.h

Change 43415 on 2002/07/31 by efong@efong_crayola_linux_cvd

changed from chip/test_regs.h to ar_test/test_regs.h

Change 43414 on 2002/07/31 by efong@efong_crayola_linux_cvd

changed from chip/test_regs.h to ar_test/test_regs.h

Change 43413 on 2002/07/31 by efong@efong_crayola_linux_cvd

switched from chip/test_regs.h to ar_test/test_regs.h

Change 43412 on 2002/07/31 by efong@efong_crayola_linux_cvd

changed from chip/test_regs.h to ar_test/test_regs.h

Change 43411 on 2002/07/31 by efong@efong_crayola_linux_cvd

switched from chip/test_regs.h to ar_test/test_regs.h

Change 43410 on 2002/07/31 by efong@efong_crayola_linux_cvd

changed from chip/test_regs.h to ar_test/test_regs.h

Change 43408 on 2002/07/31 by efong@efong_crayola_linux_cvd

switched from chip/test_regs.h to ar_test/test_regs.h

Change 43407 on 2002/07/31 by efong@efong_crayola_linux_cvd

switched from chip/test_regs.h to ar_test/test_regs.h

Change 43406 on 2002/07/31 by efong@efong_crayola_linux_cvd

changed from chip/test_regs.h to ar_test/test_regs.h

Change 43405 on 2002/07/31 by efong@efong_crayola_linux_cvd

changed from chip/test_regs.h to ar_test/test_regs.h

Change 43403 on 2002/07/31 by efong@efong_crayola_linux_cvd

changed from chip/test_regs.h to ar_test/test_regs.h

Change 43401 on 2002/07/31 by kevino@kevino_r400_win_marlboro

Added different texture sizes (128, 256, and 512)
Added clamping mode, but doesn't appear to work yet

Change 43398 on 2002/07/31 by efong@efong_crayola_linux_cvd

changed from chip/test_regs.h to ar_test/test_regs.h

Change 43348 on 2002/07/31 by kevino@kevino_r400_sun_marlboro

Resized images of the dilbert.ppm (128x128) to have larger texture maps to work with

Change 43283 on 2002/07/30 by abeaudin@abeaudin_r400_win_marlboro

multisample and jitter stuff

Change 43263 on 2002/07/30 by abeaudin@abeaudin_r400_win_marlboro

new test for jitter

Change 43239 on 2002/07/30 by georgev@georgev_r400_linux_marlboro

Fixed depth pixel problem.

Change 43230 on 2002/07/30 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 43229 on 2002/07/30 by mkelly@fl_mkelly_r400_win_laptop

Line stipple variations...

Change 43200 on 2002/07/30 by ygiang@ygiang_r400_win_marlboro_p4

fixed: SP pixel shader tests name and test cases.

Change 43183 on 2002/07/30 by llefebvr@llefebvre_laptop_r400_emu

Tests for ifs, loops and relative addressing

Change 43095 on 2002/07/30 by smoss@smoss_crayola_win

SU tests

Change 42949 on 2002/07/29 by mkelly@fl_mkelly_r400_win_laptop

128 packets, one triangle per packet, each hitting a subsample
in one JSS pixel. For each JSS_SAMPLE_SEL, each of 16 JSS pixels
are tested. JSS_SAMPLE_SEL is cycled from from 0 to 8. This test
ensures the tested JSS_SAMPLE_SEL is a unique value when compared
to all non-tested JSS_SAMPLE_SELs.

Change 42888 on 2002/07/29 by fhsien@fhsien_r400_linux_marlboro

Set Alpha to 1.0

Change 42887 on 2002/07/29 by frivas@FL_FRivas

Update to screen offset position and spacing between polygons.

Change 42882 on 2002/07/29 by mkelly@fl_mkelly_r400_win_laptop

Update tests now that zbuffer functionality is working in the RB...

Change 42825 on 2002/07/28 by abeaudin@abeaudin_r400_win_marlboro

start of changes for doing multisampling

Change 42788 on 2002/07/26 by csampayo@fl_csampayo_r400

1. Added 2 new SU tests
2. Updated test_list and tracker, accordingly

Change 42770 on 2002/07/26 by fhsien@fhsien2_r400_win_marlboro

Update Stencil fail tests

Change 42657 on 2002/07/26 by georgev@georgev_r400_linux_marlboro

Added loop tests and support.

Change 42631 on 2002/07/26 by ctaylor@fl_ctaylor_r400_dtwin_marlboro

Changed MSAA_NUM_SAMPLES to be a 3-bit field (instead of 4).
Fix bug in sc_sample_mask_in dump for jss_sample_sel field
Fix bug in sc_samplemask test bench for jss_sample_sel field
Update sc tests which set num_samples to 8.

Change 42562 on 2002/07/25 by omesh@ma_omesh

Seperated some test conditions for testing.

Change 42530 on 2002/07/25 by ashishs@fl_ashishs_r400_win

CL TESTS: testing guard band clipping of edgeflags.

Change 42512 on 2002/07/25 by frivas@FL_FRivas

Added two HOS tests for precision.

Change 42510 on 2002/07/25 by frivas@FL_FRivas

Initial check in of HOS precision tests. Tessellation and reuse vary from 1-14 and 4-16, respectively.

Change 42505 on 2002/07/25 by mkelly@fl_mkelly_r400_win_laptop

tests..

Change 42499 on 2002/07/25 by mkelly@fl_mkelly_r400_win_laptop

Basic MSAA functionality tests...

Change 42340 on 2002/07/25 by frivas@FL_FRivas

Added two HOS auto index tests (line list and triangle list).

Change 42322 on 2002/07/25 by smoss@smoss_crayola_win

Modified for Unix - watch out for test output path!

Change 42253 on 2002/07/24 by ygiang@ygiang_r400_win_marlboro_p4

removed: Changing tests name...

Change 42251 on 2002/07/24 by mkelly@fl_mkelly_r400_win_laptop

Reversed the time stamp on the output directory name
to help reading multiple directories by eye easier

Change 42248 on 2002/07/24 by frivas@FL_FRivas

Initial check-in of HOS auto index tests.

Change 42247 on 2002/07/24 by ygiang@ygiang_r400_win_marlboro_p4

fixed: const VAR

Change 42234 on 2002/07/24 by mkelly@fl_mkelly_r400_win_laptop

* Update to JSS tests
* New MSAA test verifying all 8 subsamples in basic hit tests
* Update tracker
* Update SC test_list

Change 42207 on 2002/07/24 by fhsien@fhsien2_r400_win_marlboro

ADD stencil fail tests

Change 42202 on 2002/07/24 by fhsien@fhsien2_r400_win_marlboro

Add color tiling

Change 42137 on 2002/07/24 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 42134 on 2002/07/24 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 42121 on 2002/07/24 by mkelly@fl_mkelly_r400_win_laptop

delete

Change 42081 on 2002/07/23 by ashishs@fl_ashishs_r400_win

update

Change 42052 on 2002/07/23 by frivas@FL_FRivas

Update. Removed call to "Set_VGT_INDEX_OFFSET_index_offset" from code because of an update to the register set.

Change 42030 on 2002/07/23 by csampayo@fl_csampayo2_r400

Removed function Set_VGT_INDEX_OFFSET_index_offset,as per, latest Primlib change

Change 42021 on 2002/07/23 by csampayo@fl_csampayo2_r400

Reverted change made for setting INDX_OFFSET

Change 42017 on 2002/07/23 by mkelly@fl_mkelly_r400_win_laptop

AA_MASK and MAX_SAMPLE_DISTANCE basic functionality

Change 42015 on 2002/07/23 by ashishs@fl_ashishs_r400_win

original test r400cl_frustum_edgeflags_01 having all combination of frustum plane clipping for edgeflags removed.
Made individual tests for each frustum plane (total 6 tests) clipping edgeflag combinations.

Change 42010 on 2002/07/23 by smoss@smoss_crayola_win

update

Change 42005 on 2002/07/23 by abeaudin@abeaudin_r400_win_marlboro

new rb test for multisampling

Change 41986 on 2002/07/23 by georgev@georgev_r400_linux_marlboro

No Change.

Change 41984 on 2002/07/23 by georgev@ma_georgev

No change for build test.

Change 41983 on 2002/07/23 by smoss@smoss_crayola_win

SU tests

Change 41939 on 2002/07/23 by ashishs@fl_ashishs_r400_win

To test guard band clipping for edgeflags
The test divides the display area in four equal planes(top,bottom,right,left)
Each plane has a specified number packets and each packet has a different setting in accordance to edgeflag and indexing. The number of test cases in a plane can be modified and also the display area for each case is equally divided.
Test parameters that can be easily controlled are :
1. DISP_X_DIM and DISP_Y_DIM
2. NUMBER OF PLANES IN X AND Y DIRECTION
3. NUMBER OF TEST CASES IN EACH PLANE

Change 41905 on 2002/07/23 by ashishs@fl_ashishs_r400_win

update

Change 41898 on 2002/07/23 by smoss@smoss_crayola_win

shortened tests

Change 41849 on 2002/07/22 by ashishs@fl_ashishs_r400_win

To test frustum clipping for edgeflags
The test divides the display area in six equal planes(near,far,top,bottom,right,left)
Each plane has a specified number packets and each packet has a different setting in accordance to edgeflag and indexing. The number of test cases in a plane can be modified and also the display area for each case is equally divided.
Test parameters that can be easily controlled are :
1. DISP_X_DIM and DISP_Y_DIM
2. NUMBER OF PLANES IN X AND Y DIRECTION
3. NUMBER OF TEST CASES IN EACH PLANE

Change 41822 on 2002/07/22 by mkelly@fl_mkelly_r400_win_laptop

* Basic JSS functional coverage
* Update test list
* Update tracker
* Added Plan vs. Actual graph to tracker sheet "schedule"

Change 41820 on 2002/07/22 by georgev@georgev_r400_linux_marlboro

Partial changes.

Change 41730 on 2002/07/22 by mkelly@fl_mkelly_r400_win_laptop

Improve test description and conveyance of test purpose in the image

Change 41701 on 2002/07/21 by smoss@smoss_crayola_win

update golds

Change 41624 on 2002/07/19 by csampayo@fl_csampayo2_r400

Golds for updated image size

Change 41622 on 2002/07/19 by mkelly@fl_mkelly_r400_win_laptop

Update..

Change 41620 on 2002/07/19 by mkelly@fl_mkelly_r400_win_laptop

1. Re-wrote JSS test to match latest SC spec.
2. Update test_list
3. Update tracker

Change 41606 on 2002/07/19 by frivas@FL_FRivas

update to framebuffer size

Change 41596 on 2002/07/19 by llefebvr@llefebvre_laptop_r400_emu

Corrected the VGT->SQ event interface. Corrected the GFX_COPY_STATE problem.

Change 41591 on 2002/07/19 by fhsien@fhsien2_r400_win_marlboro

working Stencil tests

Change 41590 on 2002/07/19 by csampayo@fl_csampayo_r400

Updated the following 2 tests for: switch case selector and image size respectively

Change 41570 on 2002/07/19 by frivas@FL_FRivas

Update to size of output image.

Change 41563 on 2002/07/19 by frivas@FL_FRivas

Adjusted size of output image and spacing of objects inside the image.

Change 41550 on 2002/07/19 by smoss@smoss_crayola_win

new x,y dim

Change 41547 on 2002/07/19 by fhsien@fhsien2_r400_win_marlboro

Update Stencil test

Change 41545 on 2002/07/19 by frivas@FL_FRivas

Decreased the size of the output image from 256x256 to 128x128 and adjusted the spacing of the objects being drawn.

Change 41488 on 2002/07/19 by mkelly@fl_mkelly_r400_win

Enable Window Offset Enable functionality in VTE

Change 41449 on 2002/07/19 by vgoel@fl_vgoel2

test file names are changes and hence old files are deleted

Change 41446 on 2002/07/19 by fhsien@fhsien2_r400_win_marlboro

minor typo

Change 41405 on 2002/07/18 by fhsien@fhsien2_r400_win_marlboro

Update to support the new format for dump Z buffer

Change 41401 on 2002/07/18 by csampayo@fl_csampayo_r400

1. Added 4 new SU edgeflag tests
2. Updated framebuffer size for 2 tests
3. Updated test_list and test tracker accordingly

Change 41369 on 2002/07/18 by fhsien@fhsien2_r400_win_marlboro

still trying to debug

Change 41367 on 2002/07/18 by kevino@kevino_r400_linux_marlboro

Changed to only draw 64 pixels from (0,0) to (7,7) since SSQ can't do more than that yet.

Change 41278 on 2002/07/18 by mkelly@fl_mkelly_r400_win_laptop

Update to SU SC spec.

Change 41265 on 2002/07/18 by fhsien@fhsien2_r400_win_marlboro

Change Z buffer size to 32

Change 41250 on 2002/07/18 by georgev@georgev_r400_linux_marlboro

Turned z buffer down.

Change 41205 on 2002/07/18 by mkelly@fl_mkelly_r400_win

PA dumps on/off control for full regression scripting...

Change 41201 on 2002/07/18 by mkelly@fl_mkelly_r400_win_laptop

When .mem_framebuf_r image is enabled for compare,
a bit for bit compare takes place. The headers are ignored.

Tolerance adjustment is an option. Defaults to zero tolerance.

Useage: regress_r400 -d <0-255>

If the difference is greater than the tolerance, then compare fails.

If tolerance is exceeded, difference is reported to "compare.log"
at the output destination root directory, "$/r400_regress/<user_timestamp>".

Change 41183 on 2002/07/17 by csampayo@fl_csampayo_r400

1. Added 2 new tests to SU
2. Updated SU test_list
3. Updated test tracker accordingly

Change 41146 on 2002/07/17 by mkelly@fl_mkelly_r400_win_laptop

Test in progress...

Change 41139 on 2002/07/17 by georgev@georgev_r400_linux_marlboro

Added new features.

Change 41112 on 2002/07/17 by smoss@smoss_crayola_win

SU Tests

Change 41092 on 2002/07/17 by ygiang@ygiang_r400_win_marlboro_p4

added: sp test for hardware testing(add opcode)

Change 41083 on 2002/07/17 by ashishs@fl_ashishs_r400_win

update

Change 41067 on 2002/07/17 by ashishs@fl_ashishs_r400_win

This test is intended to validate the clippper processing of the edge flags for the triangle list primitive type.
Clipping with 4 UCPs is enabled and actual clipping takes place
The edge flags for each primitive are permuted accross all packets

Change 41051 on 2002/07/17 by ashishs@fl_ashishs_r400_win

CL tests updated for latest spec changes

Change 41036 on 2002/07/17 by fhsien@fhsien_r400_linux_marlboro

For RBD tests

Change 41035 on 2002/07/17 by fhsien@fhsien_r400_linux_marlboro

ADD 1st try of Stencil test

Change 41034 on 2002/07/17 by ashishs@fl_ashishs_r400_win

VTE test updated according to latest reg spec

Change 41003 on 2002/07/17 by csampayo@fl_csampayo_r400

Restore list

Change 40950 on 2002/07/16 by csampayo@fl_csampayo_r400

1. Added 2 new tests to SU
2. Updated test_list and test tracker

Change 40946 on 2002/07/16 by ashishs@fl_ashishs_r400_win

VTE tests updated according to latest register specs

Change 40939 on 2002/07/16 by ashishs@fl_ashishs_r400_win

CL tests changed according to latest register specs

Change 40916 on 2002/07/16 by frivas@FL_FRivas

Update to auto index hos tests. Still aren't done yet.

Change 40911 on 2002/07/16 by smoss@smoss_crayola_win

more converted tests

Change 40907 on 2002/07/16 by ygiang@ygiang_r400_win_marlboro_p4

added: pixel mask tests and removed unrelated test cases from existing tests

Change 40884 on 2002/07/16 by mkelly@fl_mkelly_r400_win_laptop

Added control to reset stipple pattern at beginning of each primitive in LINE_LIST.

Change 40882 on 2002/07/16 by smoss@smoss_crayola_win

Changes to polymode register and removal of serial_proc_enable

Change 40870 on 2002/07/16 by mkelly@fl_mkelly_r400_win_laptop

Force useage of DK_ROOT/bin/sh.exe to guarantee sh.exe from depot is used for "make".

Report "REGRESS: ERROR, failed to execute make command" if "make" is not succesful, but continue to next test.

Change 40869 on 2002/07/16 by csampayo@fl_csampayo_r400

1. Updated, as per, latest register spec
2. Adjusted image size

Change 40852 on 2002/07/16 by fhsien@fhsien_r400_linux_marlboro

Update tests

Change 40848 on 2002/07/16 by ygiang@ygiang_r400_win_marlboro_p4

added: needs to be debug. sp test "maskne"

Change 40845 on 2002/07/16 by lseiler@lseiler_r400_win_marlboro2

Changed Z from 5.0 to 0.5

Change 40844 on 2002/07/16 by mkelly@fl_mkelly_r400_win_laptop

bugs to debugify

Change 40843 on 2002/07/16 by csampayo@fl_csampayo_r400

Updated packet geometries

Change 40841 on 2002/07/16 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 40792 on 2002/07/15 by fhsien@fhsien_r400_linux_marlboro

z24 bit version of the z function test.

Change 40771 on 2002/07/15 by csampayo@fl_csampayo_r400

1. Added SU test to validate provoking vertex
2. Updated SU's test_list accordingly
3. Updated the VGT test r400vgt_provoking_vtx_all_01
4. Updated test tracker with status for the test: r400su_provoking_vtx_rectangle_01

Change 40743 on 2002/07/15 by frivas@FL_FRivas

Initial check in of tests that do tessellation on auto-index primitives.

Change 40707 on 2002/07/15 by fhsien@fhsien_r400_linux_marlboro

Change r400rb_z_functions to r400rb_z16_functions

Change 40705 on 2002/07/15 by fhsien@fhsien_r400_linux_marlboro

TkP4 - RENAME

Change 40704 on 2002/07/15 by fhsien@fhsien_r400_linux_marlboro

1st release version of the basic 16 bit Z function test

Change 40703 on 2002/07/15 by mkelly@fl_mkelly_r400_win_laptop

Update for new spec.

Change 40701 on 2002/07/15 by mkelly@fl_mkelly_r400_win_laptop

Update for new spec.

Change 40699 on 2002/07/15 by mkelly@fl_mkelly_r400_win_laptop

Update to new spec.

Change 40677 on 2002/07/15 by abeaudin@abeaudin_r400_win_marlboro

added rop3 and color format support to the RB

Change 40663 on 2002/07/15 by mkelly@fl_mkelly_r400_win_laptop

Spec. Update...

Change 40649 on 2002/07/15 by mkelly@fl_mkelly_r400_win_laptop

Spec. Update...

Change 40625 on 2002/07/15 by frivas@FL_FRivas

updated shaders to version 2.0

Change 40579 on 2002/07/15 by mkelly@fl_mkelly_r400_win_laptop

Update with new pa dump control

Change 40551 on 2002/07/15 by mkelly@fl_mkelly_r400_win_laptop

Update to latest spec.

Change 40477 on 2002/07/14 by fhsien@fhsien_r400_linux_marlboro

Update for save keeping

Change 40222 on 2002/07/12 by ctaylor@fl_ctaylor_r400_win_marlboro

Fix bug in SC with RECTANGLE_LIST reference vertex assignment
Change line_stipple auto_reset_enable field to 2-bit auto_reset_cntl field.
Regoldenize r400vgt_index_size_01 results as RECTANGLE_LIST fix affected this test's results.

Change 40140 on 2002/07/12 by csampayo@fl_csampayo2_r400

Updated per latest register spec

Change 40123 on 2002/07/12 by frivas@FL_FRivas

Changed display size of image to 256x256.

Change 40107 on 2002/07/12 by csampayo@fl_csampayo2_r400

Updated per latest register spec

Change 40102 on 2002/07/12 by csampayo@fl_csampayo2_r400

Updated for swap correction.

Change 40095 on 2002/07/12 by mkelly@fl_mkelly_r400_win_laptop

Update due to spec. change...

Change 40076 on 2002/07/12 by llefebvr@llefebvre_laptop_r400_emu

again some Z related fixes in SQ tests

Change 40073 on 2002/07/12 by frivas@FL_FRivas

Updates to code to reflect changes made to the PA/SU/SC register sets.

Change 40051 on 2002/07/12 by mkelly@fl_mkelly_r400_win_laptop

Update due to spec. change...

Change 40031 on 2002/07/12 by csampayo@fl_csampayo2_r400

Updated to correct swap modes

Change 40030 on 2002/07/12 by kevino@kevino_r400_win_marlboro

Only have w !=1.0 at one corner

Change 40027 on 2002/07/12 by kevino@kevino_r400_win_marlboro

Just another try at PC

Change 40012 on 2002/07/12 by llefebvr@llefebvre_laptop_r400_emu

corrected sq tests to enable z writes...

Change 40008 on 2002/07/12 by mkelly@fl_mkelly_r400_win_laptop

Update for new spec. change

Change 40005 on 2002/07/12 by kevino@kevino_r400_win_marlboro

make shader filenames strings throughout.
update shader program to (hopefully) mult ST by 1/W

Change 39999 on 2002/07/12 by kevino@kevino_r400_win_marlboro

Updated the test to include a couple of basic mip tests and a 1st attempt at a persp correction test.
Added shader progs for these

Change 39986 on 2002/07/12 by llefebvr@llefebvre_laptop_r400_emu

enable z write enable...

Change 39957 on 2002/07/12 by llefebvr@llefebvre_laptop_r400_emu

new simple objects for SQ testing.

Change 39944 on 2002/07/12 by llefebvr@llefebvre_laptop_r400_emu

corrected tests to include Orlando's changes to the PA/SC registers.

Change 39867 on 2002/07/11 by kryan@kryan_r400_win_marlboro

- Fixed typo in CHANNEL_B_SHIFT for little endian platforms

causing compilation error on these platforms.

- Also moved all platform specific constants to top of file

in one place.

- Updated PrimLib header include files to use new

primlib_test_include.h file.

Change 39864 on 2002/07/11 by kryan@kryan_r400_sun_marlboro

Made edits to test to try to resolve the differences between the golden
images and the new dump files created.

Made changes to platform dependent values based on endianess of
platform architecture. Namely this was the texture color uint32 definitions,
the channel mask definitions, and the channel shift values which depend
on the platform architecture.

*** Note this resolved most of the differences between the new Emulator
images and the golden images. But there are still some small differences
of +/- 2 in some of the channels for a few pixels. I will leave this to the
original test author to further debug these small differences which might
be easier if some of the features of the test can be disabled.

Change 39835 on 2002/07/11 by omesh@ma_omesh

Added 11 testcases for the Color Combination testing. Visually verified the result of 5 testcases (non random cases).
Also corrected a typo in a comment in the earlier Color Destination tests.

Change 39796 on 2002/07/11 by csampayo@fl_csampayo_lt_r400

Updated per latest register definitions

Change 39792 on 2002/07/11 by omesh@ma_omesh

Added the Color Destination path tests which are almost a mirror of the Color Source path tests. These are another 31 testcases, 16 of which are randomized versions. I have visually verified the results of the non-randomized testcases (15 testcases).
Also made a minor bug fix to the earlier Color Source path tests to include Alpha blending programming of the destination path (Destination Alpha is also used in some Color Source blending modes)
Also cleaned up code and added comments.

Change 39781 on 2002/07/11 by mkelly@fl_mkelly_r400_win_laptop

Update for new SC/SU blk changes

Change 39777 on 2002/07/11 by jhoule@jhoule_r400_win_marlboro

Moved functions into a TUtil class (tutils.{h|cpp}).
Updated Crayel to have utility functions for setting conversion functions (easier than specifying all the field stuff).
Updated Makefile (new files added, old files removed).
Added TPBlender class to isolate blending operations.
Modified TexturePipe class to use it (workaround for weird dynamic alloc of TPBlender, ended up using compiler-allocated instance instead of using new).

r400tp_test.cpp: easier texture load modif, activated bilin mag by default.

Change 39764 on 2002/07/11 by georgev@ma_georgev

Fixed registers.

Change 39741 on 2002/07/11 by abeaudin@abeaudin_r400_win_marlboro

new tests for color blending

Change 39738 on 2002/07/11 by abeaudin@abeaudin_r400_win_marlboro

blender format tests

Change 39730 on 2002/07/11 by llefebvr@llefebvre_laptop_r400_emu

Fixed the cnde test to acheive the correct behaviour.

Change 39724 on 2002/07/11 by georgev@georgev_r400_linux_marlboro

Added negated predicates.

Change 39723 on 2002/07/11 by abeaudin@abeaudin_r400_win_marlboro

blender format test for uint8

Change 39719 on 2002/07/11 by ygiang@ygiang_r400_win_marlboro_p4

added: shadder test needs to be debug

Change 39699 on 2002/07/11 by llefebvr@llefebvre_laptop_r400_emu

simple obj is not rendering anything.

Change 39688 on 2002/07/11 by abeaudin@abeaudin_r400_win_marlboro

new pixel shader for unsigned 8 bit int

Change 39681 on 2002/07/11 by llefebvr@llefebvre_laptop_r400_emu

corrected some of the SQ tests to use Zbuffering

Change 39669 on 2002/07/11 by georgev@ma_georgev

Fixed changed registers.

Change 39666 on 2002/07/11 by georgev@georgev_r400_linux_marlboro

Fixed bad changed RB registers.

Change 39638 on 2002/07/11 by smoss@smoss_crayola_win

New tests

Change 39577 on 2002/07/10 by smoss@smoss_crayola_win

changed xy

Change 39559 on 2002/07/10 by ctaylor@fl_ctaylor_r400_win_marlboro

Changed PA/SU/SC register sets as follows:
A substantial number of changes has been made to the R400 PA/SU/SC register set in response to
1) Requested additions of functionality (2nd scissor rect, cliprect enable, window_offset_enable, etc)
2) Rearrangement of register fields for better driver usage
3) Architecture changes to the Antialiasing POR for R400

The register changes are as follows:
The cases where there are new or deleted registers are highlighted in bold.
REMOVED PA_SC_MSAA_2X2_OFFSET as this functionality no longer supported under new R400 AA plan.
REMOVED PA_SC_AA_OFFSET_TBL_0-3 as this functionality has changed under new R400 AA plan.
ADDED PA_SC_AA_JSS_SAMPLE_SEL_0 & 1 as this functionality has changed under new R400 AA plan.

Removed PATTERN_START field from PA_SC_LINE_STIPPLE as this is redundant with CURRENT_PTR in PA_SC_LINE_STIPPLE_STATE
Moved PA_SC_LINE_STIPPLE_STATE outside of GFXDEC space as this is now a non-context register (requires flush to read/write).
Removed DRAW_ZERO_LENGTH_LINE field from PA_SC_LINE_CNTL as this was added to support what has since been determined to be a bug in R100-R300.
REMOVED PA_SC_CNTL register.
Moved PA_SC_CNTL.OUTPUT_SCREEN_XY to SQ_CONTEXT_MISC register.
Moved 2-bit PROVOKING_VTX field from SQ_PROVOKING_VTX to 1-bit (FIRST vs. LAST) field in PA_SU_SC_MODE_CNTL
REMOVED SQ_PROVOKING_VTX as the sole field in the register was moved.
REMOVED PA_SU_POLY_OFFSET_ENABLE, PA_SU_POLY_MODE, PA_SU_CULL_MODE, all fields moved to PA_SU_SC_MODE_CNTL
ADDED PA_SU_SC_MODE_CNTL as above.
Moved PERSP_CORR_DIS field from PA_SU_VTX_CNTL to PA_SU_SC_MODE_CNTL
Moved MSAA_ENABLE from PA_SC_AA_CONFIG to PA_SU_SC_MODE_CNTL as it may be more frequently changing.
Removed Line_AA_Enable and Point_AA_Enable fields from PA_SC_AA_CONFIG as this can be controlled via MSAA_ENABLE field.
BOUNDARY_EDGE_FLAG_ENA marked as unused, but preserved pending further investigation.
Removed SERIAL_PROC_ENA bit from PA_CL_VTE_CNTL as this bit no longer functional.
Added VTX_KILL_OR field to PA_CL_CLIP_CNTL register for Vertex Shader Export Kill Flag Control.
Moved LINE_STIPPLE_ENABLE from PA_SC_CNTL to PA_SU_SC_MODE_CNTL
Removed SCISSOR_ENABLE as the scissors can be disabled by setting xmin,ymin to 0 and xmax,ymax to 8K.
Added VTX_WINDOW_OFFSET_ENABLE, and CLIPRECT_ENABLE to PA_SU_SC_MODE_CNTL
ADDED PA_SC_SCREEN_SCISSOR_TL / BR registers. Moved PA_SC_CLIPRECT_* down in address space to accomodate in SubBlock B
Renamed PA_SC_SCISSOR_* to PA_SC_WINDOW_SCISSOR as this scissor will have window offset conditionally added.
Added WINDOW_OFFSET_DISABLE to PA_SC_WINDOW_SCISSOR register to allow non-offset scissor.
Moved EXPAND_LINE_WIDTH field from PA_SC_AA_CONFIG to PA_SC_LINE_CNTL.

Change 39555 on 2002/07/10 by omesh@ma_omesh

Added the random test cases for each standard test case and also added an overall random case for the entire series in this file.

Change 39512 on 2002/07/10 by ygiang@ygiang_r400_win_marlboro_p4

fixed: sp test name

Change 39508 on 2002/07/10 by ygiang@ygiang_r400_win_marlboro_p4

added: pixel shader tests

Change 39504 on 2002/07/10 by omesh@ma_omesh

Added the limit testing version of the Color Source blending tests. Visually verified results of all 15 tests. One of the modes, GL_ONE_MINUS_DST_COLOR seems to render a suspiciously colored triangle, but I will later confirm if the result is wrong.

In these tests, need to:
1) Clean up and comment code.
2) Add randomized test case versions of these tests, to yield another 15 test cases.

Change 39467 on 2002/07/10 by mkelly@fl_mkelly_r400_win

Update with dumps for clock sim

Change 39440 on 2002/07/10 by georgev@georgev_r400_linux_marlboro

Followed new syntax of predicate "P =" to register.

Change 39435 on 2002/07/10 by mkelly@fl_mkelly_r400_win_laptop

Re-wrote test to include texture...

Change 39379 on 2002/07/10 by georgev@ma_georgev

First revision in new directory.

Change 39376 on 2002/07/10 by georgev@ma_georgev

No change.

Change 39375 on 2002/07/10 by georgev@ma_georgev

Before changes.

Change 39373 on 2002/07/10 by georgev@ma_georgev

Moved to a new directory.

Change 39347 on 2002/07/10 by smoss@smoss_crayola_win

SU tests

Change 39331 on 2002/07/10 by smoss@smoss_crayola_win

SU Tests

Change 39299 on 2002/07/10 by mkelly@fl_mkelly_r400_win_laptop

Log for investigation...

Change 39220 on 2002/07/09 by csampayo@fl_csampayo_r400

1. Added 1 test to SU suite
2. Updated test_list accordingly
3. Updated relevant status on the test tracker

Change 39160 on 2002/07/09 by fhsien@fhsien_r400_linux_marlboro

Add the r400rb_z_functions

Change 39158 on 2002/07/09 by fhsien@fhsien_r400_linux_marlboro

Second version of the test

Change 39124 on 2002/07/09 by ygiang@ygiang_r400_win_marlboro_p4

added: shader test for debug

Change 39084 on 2002/07/09 by csampayo@fl_csampayo_r400

1. Added 7 new VGT edge flags tests
2. Updated test_list accordingly

Change 39079 on 2002/07/09 by frivas@FL_FRivas

Update to starting tessellation level (changed from 0 to 1).

Change 39062 on 2002/07/09 by mkelly@fl_mkelly_r400_win_laptop

VFD Edge flag support for Constant to EX1 operation with example test.

Change 39007 on 2002/07/09 by mkelly@fl_mkelly_r400_win

Update

Change 39006 on 2002/07/09 by mkelly@fl_mkelly_r400_win

Update with latest tests

Change 38991 on 2002/07/09 by smoss@smoss_crayola_win

SU Golds

Change 38987 on 2002/07/09 by smoss@smoss_crayola_win

updated gold

Change 38786 on 2002/07/08 by csampayo@fl_csampayo_r400

1. Adding 3 SU provoking vertex tests.
2. Update test_list accordingly

Change 38762 on 2002/07/08 by frivas@FL_FRivas

Update to starting tessellation level (changed from 0.99 to 1.99)

Change 38753 on 2002/07/08 by abeaudin@abeaudin_r400_win_marlboro

added 28 rop3 functions to blender

Change 38653 on 2002/07/08 by mkelly@fl_mkelly_r400_win_laptop

Update to sanity tests for including in full regression...

Change 38646 on 2002/07/08 by mkelly@fl_mkelly_r400_win_laptop

VFD Edge Flag implementation example...

Change 38415 on 2002/07/05 by omesh@ma_omesh

Added the same original vertex and pixel shader (that doesn't process alpha) that originally existed, so Frank can use this in his Z tests and ignore all alpha related programming in his tests.

Change 38347 on 2002/07/05 by mkelly@fl_mkelly_r400_win_laptop

VFD edge flag support and example, note: waiting on bug fix in ccgen.cpp to be checked in.

Change 38260 on 2002/07/05 by dclifton@dclifton_r400

Added clippabc.dmp

Change 38247 on 2002/07/05 by mkelly@fl_mkelly_r400_win

SU test list

Change 37893 on 2002/07/03 by mkelly@fl_mkelly_r400_win_laptop

Enable a third triangle...

Change 37892 on 2002/07/03 by mkelly@fl_mkelly_r400_win_laptop

Add -z option for zipping output when compare is complete, this is
really only useful for full regressions.

Change 37886 on 2002/07/03 by kevino@kevino_r400_win_marlboro

Basic texture test to hit different filtering cases.

Change 37877 on 2002/07/03 by kevino@kevino_r400_linux_marlboro

Just another ppm to play around with as a texture

Change 37866 on 2002/07/03 by omesh@ma_omesh

Since destination (color/alpha) programming tests are a mirror of the source (color/alpha) programming tests, forgot to also mirror the input numbers to the muxes corresponding to the mirrored blending modes. Fixed this in these tests and verified result visually with the emulator.

Change 37852 on 2002/07/03 by mkelly@fl_mkelly_r400_win_laptop

Good test for basic checks of sequence/scan conversion/shader operation...

Change 37824 on 2002/07/03 by smoss@smoss_crayola_win

SU Golds

Change 37814 on 2002/07/03 by mdoggett@mdoggett_r400_linux_marlboro

Texture Cache simple texture and dxtc1 tests

Change 37783 on 2002/07/03 by mkelly@fl_mkelly_r400_win_laptop

Update comments

Change 37782 on 2002/07/03 by mkelly@fl_mkelly_r400_win_laptop

SC Baryc tests...

Change 37753 on 2002/07/02 by csampayo@fl_csampayo2_r400

1. Changed vertex data so, that, each vertex has a different color
2. Changed display settings

Change 37742 on 2002/07/02 by csampayo@fl_csampayo_r400

Updates:
1. Updated description/status for the following tests in the tracker:
r400vgt_provoking_vtx_all_01 r400vgt_hos_cubic_pos_pnt_discrete_01
2. Sorted test_list

Change 37703 on 2002/07/02 by csampayo@fl_csampayo2_r400

Updated test as follows:
1. Changed tessellation range to 1 thru 14
2. Changed number of cases to 14
3. Change packet scale and rotation

Change 37615 on 2002/07/02 by ashishs@fl_ashishs_r400_win

updated for pitch%32

Change 37611 on 2002/07/02 by frivas@FL_FRivas

Added two new HOS PNL tests

Change 37604 on 2002/07/02 by frivas@FL_FRivas

Update to number of registers vertex shader uses.

Change 37575 on 2002/07/02 by mkelly@fl_mkelly_r400_win_laptop

Added option -s which will find your current sync and stamp it on the output directory. By default, it will not determine your latest client sync.

In the default mode, the output directory will be stamped by user name in the format <user>_<time_stamp>.

If -s is used, the output directory will be in the format <sync#>_<time_stamp>.

Script will die in full regression mode if test_list does not exist.

Format output files slightly different to work with Internal web scripts.

Change 37553 on 2002/07/02 by fhsien@fhsien_r400_linux_marlboro

First rev. of the Z tests.

Change 37502 on 2002/07/01 by jhoule@jhoule_r400_win_marlboro

Simple interface to texture loading problem.
Created TEXTURE_MANAGER class for Primlib.
Update r400tp_test.cpp to call the new routines.
Added deallocate to UberChain.
Changed default format from INT to RF in TFetchConst.
NOTE: delayed change of tiling for now (broke regression).

Change 37479 on 2002/07/01 by csampayo@fl_csampayo_r400

Deleted test since, no longer valid

Change 37477 on 2002/07/01 by csampayo@fl_csampayo_r400

1. Updated the display size, test description and other minor test format changes for the following tests:
r400vgt_dma_swap_indx16_01.cpp
r400vgt_dma_swap_indx32_01.cpp

2. Updated the tests descriptions on the test tracker for above tests

3. Deleted the following test from the test tracker and adjusted
Schedule accordingly:
r400vgt_hos_PNQ_lp_ln_cont_13_16_texture_lighting_projection.cpp

Change 37466 on 2002/07/01 by csampayo@fl_csampayo_r400

Regoldenize test since, tessellation level had to be changed to be between 1 and 15 inclusive. Also, adjusted packet scaling (SCALE_X,SCALE_Y)

Change 37447 on 2002/07/01 by hartogs@fl_hartogs2

Added required renderstate VGT_GRP_VECT_1_CNTL to these tests.

Change 37444 on 2002/07/01 by hartogs@fl_hartogs2

Changed test to use discrete range from 1 to 14 instead of 0 to 14.

Change 37437 on 2002/07/01 by mkelly@fl_mkelly_r400_win_laptop

Temporarily commment out zbuffer tests

Change 37435 on 2002/07/01 by mkelly@fl_mkelly_r400_win_laptop

Comment out ucp combos temporarily...

Change 37434 on 2002/07/01 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 37429 on 2002/07/01 by ashishs@fl_ashishs_r400_win

update

Change 37425 on 2002/07/01 by ashishs@fl_ashishs_r400_win

VTE test to validate the VTX_W0_FMT control register .

Change 37397 on 2002/07/01 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 37396 on 2002/07/01 by mkelly@fl_mkelly_r400_win_laptop

del

Change 37387 on 2002/07/01 by hartogs@fl_hartogs

Modification to cause the VGT to check for required renderstate before using it.
If the required renderstate has never been written, then the VGT prints a message to stderr and asserts.
Added required renderstate to two tests:
r400vgt_hos_PNT_cp_qn_disc_14_04_lit_tex_proj_01.cpp
and r400vgt_hos_cubic_pos_pnt_discrete_01.cpp.
Added required renderstate VGT.VGT_VERTEX_REUSE_BLOCK_CNTL.VTX_REUSE_DEPTH to the init routine of plgx.cpp.

Change 37377 on 2002/07/01 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 37373 on 2002/07/01 by csampayo@fl_csampayo2_r400

update

Change 37372 on 2002/07/01 by mkelly@fl_mkelly_r400_win_laptop

Update comments in test.

Change 37352 on 2002/07/01 by smoss@smoss_crayola_win

Changed to 32x32

Change 37336 on 2002/07/01 by ashishs@fl_ashishs_r400_win

Update for pitch%32

Change 37333 on 2002/06/30 by ashishs@fl_ashishs_r400_win

Update for pitch%32

Change 37197 on 2002/06/28 by csampayo@fl_csampayo_r400

Updates. Some had bad index counts, others bad pix/vtx shader names, others both.

Change 37181 on 2002/06/28 by csampayo@fl_csampayo_r400

Various updates

Change 37174 on 2002/06/28 by csampayo@fl_csampayo_r400

Added the following new VGT tests and updated test tracker and test_list.

Change 37136 on 2002/06/28 by omesh@ma_omesh

Included a higher level primlib headerfile instead of the lower level ones, to clean up code.

Change 37100 on 2002/06/28 by omesh@ma_omesh

These shaders are no longer used by RB tests, as the vertex shader does not pass through alpha.... So, all RB tests are assumed to use a shader that passes through alpha, and hence these shaders are no longer needed, so I'm deleting them (after verifying that none of the *.cpp test files in this rb directory use them). The correct vertex/pixel shader pair to use is "rb_blender_emu_*.sp"

Change 37089 on 2002/06/28 by omesh@ma_omesh

Changed shader (vertex) to also pass through alpha for these tests. Also fixed some bugs. At this point I have tested almost all of the 70 testcases with the RB emulator and the results look visually correct.

Change 36955 on 2002/06/27 by omesh@ma_omesh

Added alpha programming to vertex buffer. This is obviously needed for color/alpha blending tests, but I had overlooked it earlier. I also need to modify the vertex/pixel shaders to export (and pass through) alpha from triangle vertices. The current vertex/pixel shaders only pass through R,G,B.

Change 36943 on 2002/06/27 by abeaudin@abeaudin_r400_win_marlboro

I changed the blender and there were accountable one bit differences in golden images.

Change 36941 on 2002/06/27 by abeaudin@abeaudin_r400_win_marlboro

added color and alpha blending functions to the rb

Change 36860 on 2002/06/27 by kryan@kryan_r400_win_marlboro

SURFACE

- Created new class SURFACE that is a generic surface to contain

all the properties associated with a surface such as pitch, height,

SurfaceFormat, etc.

PIXEL_SURFACE

- Created new class PIXEL_SURFACE to represent surfaces that store data on per-pixel basis. This has the necessary knowledge of how to calculate the element size and datasize for this surface needed to call Larry's address routines. Still in progress.

MEMORY_AREA

- Modified old Fill_Data() function to call new tiled Fill functions by default in LINEAR mode (only for ARGB8888 pixelTypes). This is the first step in converting tests to use the new tiled functions and have the emulator and hardware both operate in tiled mode by default. For now the Emulator will default to LINEAR mode, and the Fill_Data() routines will call the new tiled fill routines in LINEAR mode.

- Added legacy_mode argument to old Fill_Data(..., bool legacy_mode = false) to support cp_simple_triangle.cpp test that uses the Fill_Data() function to write the VertexBuffers to memory.

chip/gfx/cp/cp_simple_test.cpp

- Modified to use legacy mode of old Fill_Data() routines since it was decided that in having the old Fill_Data() functions call the new tiling fill routines, this would only be supported for ARGB8888.

Change 36838 on 2002/06/27 by csampayo@fl_csampayo_r400

Update

Change 36837 on 2002/06/27 by csampayo@fl_csampayo_r400

Added new VGT tests

Change 36836 on 2002/06/27 by mkelly@fl_mkelly_r400_win_laptop

Perspective-Correct barycentric coordinate interpolation simple tests verifying combos of ref v0 locations.

Change 36611 on 2002/06/26 by csampayo@fl_csampayo2_r400

Update

Change 36609 on 2002/06/26 by csampayo@fl_csampayo_r400

New VGT test

Change 36580 on 2002/06/26 by csampayo@fl_csampayo_r400

Update for pitch%32

Change 36577 on 2002/06/26 by smoss@smoss_crayola_win

SU tests and golds

Change 36576 on 2002/06/26 by hartogs@fl_hartogs2

Added two VGT DMA dump files.

Change 36563 on 2002/06/26 by frivas@FL_FRivas

deleted test because it is redundant now. there are other tests that are more complete.

Change 36541 on 2002/06/26 by csampayo@fl_csampayo_r400

Updated for pitch%32

Change 36540 on 2002/06/26 by csampayo@fl_csampayo2_r400

Updated and sorted.

Change 36528 on 2002/06/26 by frivas@FL_FRivas

no change

Change 36491 on 2002/06/26 by ygiang@ygiang_r400_win_marlboro_p4

updated: Makefile for sp tests

Change 36489 on 2002/06/26 by abeaudin@abeaudin_r400_win_marlboro

new shaders for blenders

Change 36369 on 2002/06/26 by smoss@smoss_crayola_win

SU Tests and gold updates

Change 36305 on 2002/06/25 by csampayo@fl_csampayo_r400

Update

Change 36303 on 2002/06/25 by csampayo@fl_csampayo_r400

Added new VGT tests

Change 36274 on 2002/06/25 by ashishs@fl_ashishs_r400_win

update

Change 36267 on 2002/06/25 by ashishs@fl_ashishs_r400_win

CL tests: Converted from R200
Vertex position combinations tested against frustum clipping for lines.

Change 36165 on 2002/06/25 by jhoule@jhoule_r400_win_marlboro

SurfaceFormat changed to ColorFormat.

Change 36116 on 2002/06/25 by frivas@FL_FRivas

Added more HOS tests to test list

Change 36081 on 2002/06/25 by mkelly@fl_mkelly_r400_win

Update

Change 36080 on 2002/06/25 by mkelly@fl_mkelly_r400_win

Bugzilla #242

Change 35950 on 2002/06/24 by csampayo@fl_csampayo_r400

Update

Change 35945 on 2002/06/24 by csampayo@fl_csampayo_r400

Adding new VGT tests

Change 35828 on 2002/06/24 by frivas@FL_FRivas

Initial check in of HOS PNQ test with a single PNQ using continuous tessellation varying between 1.0-14.0 with reuse 4-16. Lighting is implemented.

Change 35827 on 2002/06/24 by frivas@FL_FRivas

Updated descriptions in the comments and model's offset from the origin.

Change 35819 on 2002/06/24 by frivas@FL_FRivas

Initial check in of HOS PNQ test. Has 4 PNQ's that are computed with lighting and texturing using orthographic projection. Tessellation level varies between 1.99-14.99 and reuse varies between 4-16.

Change 35808 on 2002/06/24 by frivas@FL_FRivas

Initial check in of HOS PNQ test. Has 4 PNQ's that have lighting and texturing with orthographic projection. Tessellation varies from 1.0-14.0 and reuse varies from 4-16.

Change 35741 on 2002/06/24 by mkelly@fl_mkelly_r400_win_laptop

Added more dumps

Change 35645 on 2002/06/21 by ashishs@fl_ashishs_r400_win

update to test_list for CL

Change 35640 on 2002/06/21 by csampayo@fl_csampayo_r400

Update

Change 35638 on 2002/06/21 by ashishs@fl_ashishs_r400_win

CL tests: Vertex position combinations tested against frustum clipping.

Change 35618 on 2002/06/21 by frivas@FL_FRivas

Initial check in of HOS PNQ that uses continuous tessellation that varies between 1.0-14.0 and reuse 4-16 with orthographic projection. Displays as wireframe model.

Change 35617 on 2002/06/21 by frivas@FL_FRivas

update to HOS PNT tests

Change 35540 on 2002/06/21 by mkelly@fl_mkelly_r400_win_laptop

UCP combos on polymode line and line stipple

Change 35523 on 2002/06/21 by abeaudin@abeaudin_r400_win_marlboro rename test

Change 35521 on 2002/06/21 by abeaudin@abeaudin_r400_win_marlboro

blender test for emu

Change 35518 on 2002/06/21 by csampayo@fl_csampayo2_r400

Updates

Change 35508 on 2002/06/21 by mkelly@fl_mkelly_r400_win_laptop

Simple CL test

Change 35503 on 2002/06/21 by csampayo@fl_csampayo_r400

Update

Change 35490 on 2002/06/21 by mkelly@fl_mkelly_r400_win_laptop

Submit for investigation...

Change 35484 on 2002/06/21 by llefebvr@llefebvre_laptop_r400_emu

Predication optimization problem fixed in Sq.

Change 35483 on 2002/06/21 by omesh@ma_omesh

Removed all the #include "mem_class.h" lines. Looks like that header file doesn't exist anymore and was absorbed into some other headerfile, as the test code compiles and runs even without this.

Change 35474 on 2002/06/21 by mkelly@fl_mkelly_r400_win_laptop

Fix includes so it will build

Change 35358 on 2002/06/20 by mkelly@fl_mkelly_r400_win

Update, removed "debug" text from emulate time displayed at each test interval. Enabled extra options for web publishing.

Change 35337 on 2002/06/20 by ashishs@fl_ashishs_r400_win

update to test_list file for CL and VTE

Change 35317 on 2002/06/20 by mkelly@fl_mkelly_r400_win_laptop

Basic stipple triangle and polygon tests

Change 35313 on 2002/06/20 by llefebvr@llefebvre_laptop_r400_emu

Environement map tests...

Change 35288 on 2002/06/20 by ashishs@fl_ashishs_r400_win

corrected comment

Change 35245 on 2002/06/20 by ashishs@fl_ashishs_r400_win

CL tests Converted from R200.
:Vertex position combinations tested against frustum clipping.

Change 35242 on 2002/06/20 by mkelly@fl_mkelly_r400_win_laptop

adding potential bug for record

Change 35203 on 2002/06/20 by mkelly@fl_mkelly_r400_win_laptop

Update gold for clamped color export

Change 35193 on 2002/06/20 by csampayo@fl_csampayo2_r400

Update per last VFD change

Change 35177 on 2002/06/20 by omesh@ma_omesh

Added some more color/alpha combination tests that were lying around on my client.
Will later clean up all the extraneous and generic comments and add my own, more focussed functional comments.
Still have to verify that all these basic tests:
1) Program only the parameters being tested which change the output.
2) Don't leave any registers which affect the output (not being tested) to be uninitialized.... This would produce a mismatch between the simulator and emulator even for outputs we don't care about and would waste time in debugging.
Also included these 2 test files in the Makefile.

Change 35162 on 2002/06/20 by ashishs@fl_ashishs_r400_win

updated comment

Change 35154 on 2002/06/20 by ashishs@fl_ashishs_r400_win

Features Tested: clip planes
Test Purpose:creates 10 vertices from 7 clip planes

Change 35150 on 2002/06/20 by jhoule@jhoule_r400_win_marlboro

Removed old texture loading functions.

Change 35149 on 2002/06/20 by jhoule@jhoule_r400_win_marlboro

Skeleton for texture tests.
Currently does a textured quad.
Inspired from primlib_template_*, but split in functions.

Change 35105 on 2002/06/20 by llefebvr@llefebvre_laptop_r400_emu

corrected the sq tests to use the new primlib include.

Change 35078 on 2002/06/19 by csampayo@fl_csampayo2_r400

Corrected number of indices to always be at least 2.

Change 35076 on 2002/06/19 by csampayo@fl_csampayo_r400

Added new DMA specific VGT tests. Updated test_list correspondinly.

Change 35065 on 2002/06/19 by omesh@ma_omesh

Adding my locally existing (for many days!) alpha source/destination blending basic tests to the server.
These tests are 15 tests each which are mirror tests, just like the color blending tests.... With source and destination alpha programming interchanged in the 2 sets. When Alicia finishes the alpha blending in the emulator, I have asked her to inform me, so I can run these tests.

Change 34998 on 2002/06/19 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 34953 on 2002/06/19 by mkelly@fl_mkelly_r400_win_laptop

Update to include control of Z_WRITE_ENABLE and ZFUNC... awaiting support updates to RB emu_lib...

Change 34779 on 2002/06/18 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 34719 on 2002/06/18 by mkelly@fl_mkelly_r400_win_laptop

More zbuffer tests, although still need to be revisted when there is higer confidence in zbuffer functionality

Change 34713 on 2002/06/18 by frivas@FL_FRivas

Initial check in of HOS PNL tests for 4 PNL's using both cubic and linear position with orthographic projection.

Change 34712 on 2002/06/18 by frivas@FL_FRivas

delete

Change 34711 on 2002/06/18 by frivas@FL_FRivas

deleted

Change 34710 on 2002/06/18 by frivas@FL_FRivas

updates

Change 34663 on 2002/06/18 by smoss@smoss_crayola_win

fixed build error

Change 34658 on 2002/06/18 by mkelly@fl_mkelly_r400_win_laptop

SC zbuffer (RCC) interface tests

Change 34589 on 2002/06/17 by omesh@ma_omesh

Added a mirror (symmetrical test case) of r400rb_basic_color_source.cpp
Simply interchanged source and destination color/alpha programming and expecting mirrored results.

Change 34574 on 2002/06/17 by frivas@FL_FRivas

Minor update to scaling.

Change 34544 on 2002/06/17 by omesh@ma_omesh

Added a typical structure of a basic color blending test I have been playing around with for several days (and its corresponding inclusion in the Makefile). This uses 2 triangles rendered one on top of the other. The first triangle programs the destination buffer color and the second one is the real test, which programs the source triangle color (and is supposed to blend with the previous one in a specified way). I reuse most of the pointers and render state settings for the 2nd triangle from the 1st.

Change 34537 on 2002/06/17 by frivas@FL_FRivas

Initial check in of HOS PNL test. Uses 4 PNL's and does continuous tessellation at levels varying between 1.0-14.0 with reuse 4-16.

Change 34529 on 2002/06/17 by frivas@FL_FRivas

Initial check in of HOS PNL test. Uses continuous tessellation of 4 PNL's. Uses cubic position and linear normal.

Change 34520 on 2002/06/17 by ashishs@fl_ashishs_r400_win

updated the pixel centers in the test

Change 34516 on 2002/06/17 by frivas@FL_FRivas

Initial check in of HOS PNT that uses discrete tessellation and cubic postion and quadratic normal interpolation. Renders multi PNT's with lighting and texturing using orthographic projection.

Change 34510 on 2002/06/17 by frivas@FL_FRivas

Initial check in of HOS PNT test with single PNT using discrete tessellation varying from 1-14 and reuse 4-16. Uses orthographic projection.

Change 34501 on 2002/06/17 by frivas@FL_FRivas

Initial check in of HOS PNT test with single PNT using cubic position and quadratic normal interpolation with tessellation level varying between 1.0-16.0 and reuse varying between 4-16. Uses orthographic projection and lighting.

Change 34498 on 2002/06/17 by mkelly@fl_mkelly_r400_win_laptop

Test SC rectangle list vertex order interpretation...

Change 34492 on 2002/06/17 by frivas@FL_FRivas

Initial check in of HOS PNT test that uses only a single PNT in wireframe mode. Impelements lighting. Tessellation 1.0-14.0 and reuse 4-16. Uses orthographic projection.

Change 34486 on 2002/06/17 by frivas@FL_FRivas

Initial check in of a HOS multi PNT test with lighting, texturing and the ability to toggle between orthographic and perspective projection. Tessellation level varies between 1.1-14.1 and reuse varies between 4-16.

Change 34480 on 2002/06/17 by frivas@FL_FRivas

Initial check in for continuous tesselltaion PNT HOS test using lighting, and texturing with orthographic projection. Uses cubic position and quadratic normal interplation. You can toggle between orthographic and perspective projection. Tessllation ranges from 1.99-14.99 and reuse from 4-16.

Change 34455 on 2002/06/17 by frivas@FL_FRivas

Initial check in of HOS PNT test that changes the orientation of normals.

Change 34449 on 2002/06/17 by frivas@FL_FRivas

HOS test with multi PNT's that does lighting and texturing while varying both tessellation level (1.0-14.0) and reuse level (4-16). Uses continuous tessellation, cubic position and quadratic normal with orthographic projection. Perspective projection may be toggled on.

Change 34368 on 2002/06/16 by ashishs@fl_ashishs_r400_win

VTE Test:
Update (XYZ)SCALE without updating (XYZ)OFFSET between packets and vice-versa.

Change 34342 on 2002/06/15 by ashishs@fl_ashishs_r400_win

Enable (XYZ)SCALE without enabling (XYZ)OFFSET and vice-versa.
The test sends 2 packets, each packet with 2 vertex data for checking VPORT_{X|Y|Z}_SCALE_ENA registers and VPORT_{X|Y|Z}_OFFSET_ENA registers.

Change 34267 on 2002/06/14 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 34266 on 2002/06/14 by mkelly@fl_mkelly_r400_win_laptop

First line stipple tests...

Change 34201 on 2002/06/14 by mkelly@fl_mkelly_r400_win

Update

Change 34175 on 2002/06/14 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 34164 on 2002/06/14 by csampayo@fl_csampayo_r400

Golds for updated test

Change 34162 on 2002/06/14 by csampayo@fl_csampayo_r400

Update

Change 34070 on 2002/06/14 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 34064 on 2002/06/14 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 33912 on 2002/06/13 by kmahler@kmahler_r400_win_devel_views

New Mid-Level Shader Assembler Syntax - Phase I

TEST DEVELOPERS, PLEASE READ!

With the help of Steve Allen, this GNU g++ compiler error (inconsistency?) was verified using a simple "foo" program. After further investigation using "google" searches, it was determined that this error has also been seen by others. Unfortunately, the work-around is to rename (YUK) the directories to names other than "utility" and "memory".

Changing the "primlib/memory" directory to "primlib/memory_access" is a relatively easy task (PIA), but all test program Makefiles will need to be changed and/or test program source files (see methods below).

I changed all of the primlib header/source files to use relative file names in all primlib include statements. Thus, Primlib's Makefile will only specify "-I $(TESTCHIP)/primlib" to access primlib header files.

I changed all of the Makefiles that make up the regression tests: perf sys/cp gfx/sc gfx/vgt gfx/pa/su gfx/pa/cl gfx/pa/vte.

If you have tests in directories other than the ones specified above, then the simplest way to fix the tests is to change the Makefile line "-I$(PRIMLIB_DIR)/memory" to "-I$(PRIMLIB_DIR)/memory_access".

For the Tests/Makefile in the "perf" directory, I did the longer more preferred method which consists of deleting all of the "-I" options that use Primlib relative paths (keeping just "-I ($PRIMLIB_DIR)"), deleting all of the existing primlib include statements in the test program, and then including the new "primlib_test_includes.h" file (see "perf/primlib_template_simple_triangle.cpp" for details).

All NEW Makefiles and test programs should use this preferred method. You can cut-and-paste this include file from "primlib_template_simple_triangle.cpp".

As for the "test_lib/src/testchip/utility" directory. This directory contains "drand48.[h|cpp]" and a Makefile. I did a search in the TOT and could not find any reference to it. I changed the directory to "utility_lib". If any one is using this code, I apologize for any inconvenience this may cause you (send complaints to "gcc.gnu.org").

All tests pass the regression on both Winblows and Linux with the exception of "r400vgt_hos_PNT_cp_qn_disc_14_04_lit_tex_proj_01" on Linux which is a known failure.

Good Luck :)

Change 33873 on 2002/06/13 by mkelly@fl_mkelly_r400_win_laptop

back out change

Change 33870 on 2002/06/13 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 33869 on 2002/06/13 by mkelly@fl_mkelly_r400_win_laptop

Update, remove extra line from cl/test_list (even though it is ok, but just to be consistent)

Change 33842 on 2002/06/13 by csampayo@fl_csampayo2_r400

Update

Change 33782 on 2002/06/13 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 33779 on 2002/06/13 by mkelly@fl_mkelly_r400_win_laptop

VFD, clamp final color in pixel shader, support user programmability to texture fetch constant or default to using 1-8.

Change 33729 on 2002/06/13 by ashishs@fl_ashishs_r400_win

test list for cl

Change 33725 on 2002/06/13 by ashishs@fl_ashishs_r400_win

test list for vte

Change 33720 on 2002/06/13 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 33718 on 2002/06/13 by mkelly@fl_mkelly_r400_win

Update

Change 33575 on 2002/06/12 by mkelly@fl_mkelly_r400_win_laptop

checkpoint...

Change 33529 on 2002/06/12 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 33526 on 2002/06/12 by kryan@kryan_r400_win_marlboro

Updated Makefile in chip/ferret and chip/gfx/sp so that memory_area.h can find crayola_enum.h.

Change 33485 on 2002/06/12 by llefebvr@llefebvre_laptop_r400_emu

Implemented most of the remaining vector opcodes. Including pixel kills. The pred_set are still not implemented since they are still changing.

Change 33474 on 2002/06/12 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 33471 on 2002/06/12 by mkelly@fl_mkelly_r400_win_laptop

bug

Change 33466 on 2002/06/12 by mkelly@fl_mkelly_r400_win

Update

Change 33458 on 2002/06/12 by mkelly@fl_mkelly_r400_win_laptop

update comment

Change 33436 on 2002/06/12 by kryan@kryan_r400_win_marlboro

Update Makefiles to include r400/devel directory in -I include path

so that full paths can be used in include files. Memory_area.h uses

the cmn_lib/lib/address/address.h file and needs to specify the

full path.

Change 33418 on 2002/06/12 by mkelly@fl_mkelly_r400_win_laptop

Fix endian bug on unix clip regress_e

Change 33350 on 2002/06/11 by ashishs@fl_ashishs_r400_win

update

Change 33345 on 2002/06/11 by ashishs@fl_ashishs_r400_win

VTE test:
To check the VTX_W0_FMT control register .
If ON, the VTE will perform the reciprocal to get 1/W0
else indicates that the incoming W0 is not 1/W0.

Also fills gap in the VHDL coverage of the VTE by its unique combination of settings and vertex data.

Change 33329 on 2002/06/11 by mkelly@fl_mkelly_r400_win_laptop

update, sc point tests....

Change 33217 on 2002/06/11 by llefebvr@llefebvre_laptop_r400_emu

Some more stress tests for the SQ...

Change 33181 on 2002/06/11 by csampayo@fl_csampayo_r400

Udated test description

Change 33164 on 2002/06/11 by csampayo@fl_csampayo_r400

Removed .cpp from tests names

Change 33159 on 2002/06/11 by ashishs@fl_ashishs_r400_win

update

Change 33123 on 2002/06/10 by ashishs@fl_ashishs_r400_win

VTE Test:
To check Z multiply 1/W using VTX_Z_FMT = 0 To calculate the screen co-ordinates from the clip coordinates and verify:
-> see the R400 CLIP/VTE SPEC

Change 33101 on 2002/06/10 by csampayo@fl_csampayo2_r400

Update

Change 33085 on 2002/06/10 by csampayo@fl_csampayo2_r400

Update to add the following VGT test for regress_e
r400vgt_hos_PNT_cp_qn_disc_14_04_lit_tex_proj_01

Change 33081 on 2002/06/10 by csampayo@fl_csampayo2_r400

Update

Change 33080 on 2002/06/10 by csampayo@fl_csampayo2_r400

Adding VGT Tesselator tests

Change 33054 on 2002/06/10 by mkelly@fl_mkelly_r400_win_laptop

Update, sc tests...

Change 33016 on 2002/06/10 by ashishs@fl_ashishs_r400_win

VTE test:
To check XY multiply 1/W using VTX_XY_FMT = 0
To calculate the screen co-ordinates from the clip coordinates and verify:
-> see the R400 CLIP/VTE SPEC

Change 32971 on 2002/06/10 by csampayo@fl_csampayo_r400

Update

Change 32884 on 2002/06/10 by mkelly@fl_mkelly_r400_win

Update

Change 32825 on 2002/06/07 by csampayo@fl_csampayo_r400

Adding new VGT tests

Change 32808 on 2002/06/07 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 32766 on 2002/06/07 by ashishs@fl_ashishs_r400_win

To validate the vte control register by sending different number of vertices(packets) and for a set of cases change the vte control register settings and check the results.

Change 32736 on 2002/06/07 by llefebvr@llefebvre_laptop_r400_emu

Various SX->RB interface fixes. Also removed the MSB of the PS_EXPORT_MODE field since it is not needed anymore.

Change 32434 on 2002/06/06 by llefebvr@llefebvre_laptop_r400_emu

corrected dolphin test.

Change 32429 on 2002/06/06 by llefebvr@llefebvre_laptop_r400_emu

fixed the wedege test

Change 32382 on 2002/06/06 by llefebvr@llefebvre_laptop_r400_emu

the wedge model test for the SQ.

Change 32287 on 2002/06/06 by frivas@FL_FRivas

Initial check-in. HOS tests of 4 PNL's. Tessellation level = 13 and reuse level = 4.

Change 32285 on 2002/06/06 by frivas@FL_FRivas

Deleted these two tests because they have been replaced by better ones.

Change 32267 on 2002/06/06 by frivas@FL_FRivas

Initial check-in. Test for HOS using 2 PNT's with lighting, texturing, and projection. Tessellation level = 14 and reuse level = 4.

Change 32261 on 2002/06/06 by frivas@FL_FRivas

Test has been deleted because a newer and better test has been created.

Change 32251 on 2002/06/06 by frivas@FL_FRivas

Updated VTE settings to use new constant definitions instead of raw numbers.

Change 32250 on 2002/06/06 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 32235 on 2002/06/06 by smoss@smoss_crayola_win

update golds

Change 32230 on 2002/06/06 by mkelly@fl_mkelly_r400_win

Add back in frustum clipping to regress_e now that it passes :)

Change 32228 on 2002/06/06 by mkelly@fl_mkelly_r400_win

Update

Change 32165 on 2002/06/05 by csampayo@fl_csampayo_r400

Updates

Change 32157 on 2002/06/05 by mkelly@fl_mkelly_r400_win_laptop

regress_r400

Change 32126 on 2002/06/05 by csampayo@fl_csampayo_r400

Updates

Change 32107 on 2002/06/05 by csampayo@fl_csampayo_r400

Updates

Change 32070 on 2002/06/05 by llefebvr@llefebvre_laptop_r400_emu

Dolphin test

Change 32024 on 2002/06/05 by csampayo@fl_csampayo_r400

Updated tess levl from 10 to 14

Change 32022 on 2002/06/05 by csampayo@fl_csampayo_r400

Adding VGT tests

Change 31974 on 2002/06/05 by ygiang@ygiang_r400_win_marlboro_p4

added: pixel shader programs (scalar mul,min,max)

Change 31899 on 2002/06/04 by csampayo@fl_csampayo_r400

Updated per change# 31760

Change 31885 on 2002/06/04 by csampayo@fl_csampayo_r400

Updated per change# 31760

Change 31831 on 2002/06/04 by bhankins@fl_bhankins_r400_win

added files

Change 31681 on 2002/06/04 by mkelly@fl_mkelly_r400_win

Update

Change 31677 on 2002/06/04 by smoss@smoss_crayola_win

SU tests

Change 31604 on 2002/06/03 by frivas@FL_FRivas

Update to test description in source code.

Change 31603 on 2002/06/03 by frivas@FL_FRivas

Update to test description in the source code. Nothing changed in the shader.

Change 31548 on 2002/06/03 by omesh@ma_omesh

Latest version of the 7 constant color tests. They compile and run, but I can't "test the tests" as the emulator doesn't implement color blending yet.
I am still cleaning up the code for the other tests and will check them into Perforce as soon as I do that.
I have updated the Makefile to include this test too.

Change 31538 on 2002/06/03 by mkelly@fl_mkelly_r400_win_laptop

Basic x-major, y-major line verification for the SC

Change 31462 on 2002/06/03 by mkelly@fl_mkelly_r400_win_laptop

Back out test until bug is fixed.

Change 31390 on 2002/06/03 by mkelly@fl_mkelly_r400_win

Add clip frustum to regress_e

Change 31299 on 2002/05/31 by csampayo@fl_csampayo_r400

Adding tesssellator test

Change 31282 on 2002/05/31 by csampayo@fl_csampayo_r400

Update

Change 31276 on 2002/05/31 by csampayo@fl_csampayo_r400

Adding golds for regress_e

Change 31274 on 2002/05/31 by csampayo@fl_csampayo_r400

Added the following tessellator test:
r400vgt_hos_cubic_pos_pnt_discrete_01

Change 31273 on 2002/05/31 by mkelly@fl_mkelly_r400_win_laptop

Basic diamond exit tests....

Change 31270 on 2002/05/31 by csampayo@fl_csampayo_r400

Adding the following Tessellator test:

Change 31247 on 2002/05/31 by mkelly@fl_mkelly_r400_win_laptop

Test two lines in one packet, confirming hit and no hit pixels are valid.

Change 31148 on 2002/05/31 by frivas@FL_FRivas

HOS test of PNL with tessellation leve 13 using continuous tessellation. There are problems with tessellation beyond level 13 (time out issues and shader artifacts).

Change 31136 on 2002/05/31 by llefebvr@llefebvre_laptop_r400_emu

enabled Z buffering

Change 31129 on 2002/05/31 by frivas@FL_FRivas

HOS test of PNL continuous tessellation at level 13 using cubic position and linear normal interpolation. There seem to be problems at tessellation levels > 13 in that there are scan converter and time out issues.

Change 31116 on 2002/05/31 by llefebvr@llefebvre_laptop_r400_emu

Added safety checks in the sq vertex processing

Change 31069 on 2002/05/31 by frivas@FL_FRivas

HOS test of two PNT's at a discrete tessellation level of 8.0. Implements texturing, lighting, and projection.

Change 31058 on 2002/05/31 by frivas@FL_FRivas

VGT HOS tests added, modified test_list

Change 31054 on 2002/05/31 by mkelly@fl_mkelly_r400_win_laptop

Bugzilla Bug 238, RB color clamping problem

Change 30966 on 2002/05/30 by omesh@ma_omesh

Cleaned up some junk code and used new scheme for number generation based on the seperation of the TG class from the std:vector data structure.

Change 30942 on 2002/05/30 by llefebvr@llefebvre_laptop_r400_emu

updated the VGT->SQ interface (and corresponding blocks) to match HW in name and size and add the Event field.

Change 30904 on 2002/05/30 by omesh@ma_omesh

Just a basic "my own template" which is not yet a complete test. Decided on using this specific geometry for the basic tests. Will have to rethink the geometry for stress tests for maximum screen coverage of colors.
Was simply playing around with these tests and am only submiting it to Perforce to keep track of what I was playing around with.

Change 30880 on 2002/05/30 by ashishs@fl_ashishs_r400_win

corrected errors

Change 30852 on 2002/05/30 by llefebvr@llefebvre_laptop_r400_emu

parser for the obj format

Change 30849 on 2002/05/30 by ashishs@fl_ashishs_r400_win

This test is intended to validate the vertex reuse functionality with actual clipping.The test processes 56 packets each with one 84 primitive triangle list containing 16 vertices with input vertex data: XYZW0, 1 color , no textures.For each packet the test has the 252 indices making up 84 primitives, unique indices are randomly selected between 0-12 and 0-15. For each packet six UCP planes are set up so that most of the original verices get clipped.

Change 30726 on 2002/05/29 by mkelly@fl_mkelly_r400_win_laptop

Move all files in compare list to output directory irregardless of #, print warning if P4 is down.

Change 30707 on 2002/05/29 by csampayo@fl_csampayo_r400

Adding r400_regress script dependency files:

Change 30685 on 2002/05/29 by mkelly@fl_mkelly_r400_win_laptop

Widen coverage of regress_e

Change 30665 on 2002/05/29 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 30663 on 2002/05/29 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 30661 on 2002/05/29 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 30576 on 2002/05/29 by mkelly@fl_mkelly_r400_win

Update to fix Random library link error....

Change 30547 on 2002/05/29 by mkelly@fl_mkelly_r400_win

Update to regress_e, ADD 4 more SC tests

Change 30538 on 2002/05/29 by mkelly@fl_mkelly_r400_win

SC golds for regress_e

Change 30535 on 2002/05/29 by mkelly@fl_mkelly_r400_win

Update to fix random library build error.

Change 30533 on 2002/05/29 by mkelly@fl_mkelly_r400_win_laptop

Added support for HOS Cubic Position PNT with example useage in file r400vfd_vs_hos_cubic_pos_pnt_01

Change 30531 on 2002/05/29 by mkelly@fl_mkelly_r400_win_laptop

Update to remove const of INDEX_BUFFER class...

Change 30460 on 2002/05/28 by smoss@smoss_crayola_win

SU tests for regress_e

Change 30264 on 2002/05/28 by ashishs@fl_ashishs_r400_win

Tests Culling Only when UCPs are enabled, The UCPS do not clip, only cull.

Change 30248 on 2002/05/28 by ashishs@fl_ashishs_r400_win

Features Tested : user clip planes
Test Purpose:verify the 64 combinations of ucp control bits
Method: test has 64 polygons arranged in a grid of 8x8. Each polygon has a different combination of ucp enable bits set
from 0 to 63, with the upper left being 0, counting down the
columns first, then across
Expected Results: 8 rows of 8 primitives (64 total primitives), each clipped using
different combinations of the user-defined clip planes

Change 30054 on 2002/05/24 by mkelly@fl_mkelly_r400_win_laptop

Update to find tests on the client much faster...

Change 29980 on 2002/05/24 by ygiang@ygiang_r400_win_marlboro_p4

added: sp alu scalar opcode "add, min, max, mul"

Change 29976 on 2002/05/24 by mkelly@fl_mkelly_r400_win_laptop

SC tests to validate triangle coarse walk, quad evaluation, and detail pixel determination...

Change 29927 on 2002/05/24 by smoss@smoss_crayola_win

Added random libs

Change 29855 on 2002/05/24 by ygiang@ygiang_r400_win_marlboro_p4

fixed: random lib link

Change 29846 on 2002/05/24 by smoss@smoss_crayola_win

incorrect test name

Change 29765 on 2002/05/23 by ygiang@ygiang_r400_win_marlboro

cut and paste changes

Change 29764 on 2002/05/23 by smoss@smoss_crayola_win

extraneous characters in makefile

Change 29757 on 2002/05/23 by ygiang@ygiang_r400_win_marlboro

added: more tests and fail list

Change 29729 on 2002/05/23 by ygiang@ygiang_r400_win_marlboro

add: color clamp test...Clamping doesn't work yet

Change 29700 on 2002/05/23 by ygiang@ygiang_r400_win_marlboro_p4

sp test files

Change 29686 on 2002/05/23 by georgev@ma_georgev

Back integrate from devel_primlib.

Change 29613 on 2002/05/23 by smoss@smoss_crayola_win

su test

Change 29592 on 2002/05/22 by mkelly@fl_mkelly_r400_win

Update

Change 29538 on 2002/05/22 by mkelly@fl_mkelly_r400_win_laptop

Test complete.

Change 29396 on 2002/05/21 by mkelly@fl_mkelly_r400_win_laptop

Update checkpoint...

Change 29390 on 2002/05/21 by mkelly@fl_mkelly_r400_win_laptop

Example of perspective projection using the shader.

Change 29359 on 2002/05/21 by omesh@ma_omesh

Added the Makefile and shader pipe programming files needed to compile and run RB tests. The shader pipe programming files (*.SP) are taken from the simple triangle examples.

Change 29290 on 2002/05/21 by smoss@smoss_crayola_win

update to gold due to 1 bit color difference

Change 29273 on 2002/05/21 by ashishs@fl_ashishs_r400_win

corrected vertex buffer data

Change 29134 on 2002/05/20 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint

Change 29082 on 2002/05/20 by mkelly@fl_mkelly_r400_win

Update script to report build errors.

Change 29069 on 2002/05/20 by csampayo@fl_csampayo_r400

Addint VGT Tessellator test

Change 29053 on 2002/05/20 by mkelly@fl_mkelly_r400_win

Updates...

Change 28908 on 2002/05/17 by mkelly@fl_mkelly_r400_win

Update

Change 28905 on 2002/05/17 by mkelly@fl_mkelly_r400_win_laptop

* Modified to only work with files in uncommented in compare_list.
* Any file name and extension can be added to compare_list and will be used in regression for comparison to gold.
* Sync stamp on output directory is now exclusive of test_lib/src/chip/gfx

Change 28885 on 2002/05/17 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 28801 on 2002/05/17 by smoss@smoss_crayola_win

SU tests with golds

Change 28782 on 2002/05/17 by mkelly@fl_mkelly_r400_win_laptop

VFD, support for linear postion interpolation.

Change 28756 on 2002/05/17 by mkelly@fl_mkelly_r400_win_laptop

update

Change 28755 on 2002/05/17 by mkelly@fl_mkelly_r400_win_laptop

update

Change 28569 on 2002/05/16 by csampayo@fl_csampayo_r400

Adding the following VGT tests

Change 28556 on 2002/05/16 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 28551 on 2002/05/16 by mkelly@fl_mkelly_r400_win_laptop

Update gold

Change 28537 on 2002/05/16 by omesh@ma_omesh

Oops! Added the file with the wrong extension, So simply deleted the old file and added a new one with the right extension.

Here is the older comment (incase it didn't catch it from the revision history of the old file):
Added in a placeholder so that Frank and me know where to put RB tests. This is just the simple triangle test copied and pasted here.

Change 28530 on 2002/05/16 by omesh@ma_omesh

Added in a placeholder so that Frank and me know where to put RB tests. This is just the simple triangle test copied and pasted here.

Change 28397 on 2002/05/16 by ashishs@fl_ashishs_r400_win

Converted from R200, 225 gouraud shaded, clipped triangles.
Vertex position combinations tested against frustum clipping.

Change 28241 on 2002/05/15 by hwise@fl_hwise_r400_win

CP Updates:
1) Updated CP block files
2) Updated Emulator to match block file changes
3) Commented out guts of DEVICE::Init_CP() function
because it was reading the CP_STATE_IM_CNTL
register which is now uninitialized at power up
and the data being read was not being used anyway
4) Updated bmInit() function in file
devel/test_lib/src/testchip/busmaster/busmaster.cpp
because the CP interrupt is no longer on a shared
register (there should not be any shared register in
R400 now)
5) Updated gold files because the CP_STATE_IM_CNTL register
read is no longer in the *.rd_r and *.log files for the
tests that use the primlib DEVICE class

Change 28018 on 2002/05/14 by csampayo@fl_csampayo_r400

Adding VGT test

Change 27976 on 2002/05/14 by llefebvr@llefebvre_laptop_r400_emu

New SQ register map.
Added EXEC_END,CEXEC_END, CPEXEC_END. Removed the END instruction.

Change 27962 on 2002/05/14 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 27857 on 2002/05/14 by smoss@smoss_crayola_win

added dump files

Change 27853 on 2002/05/14 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint for adding HOS support to VFD. This version supports:

a. VFD_VS_HOS_CUB_POS_QUAD_NORM_PNT
b. VFD_VS_XFORM_0
c. VFD_VS_HOS_LINEAR_PNL

In this specific VFD described order. See r400vfd_hos_example_01.cpp for useage.

Change 27680 on 2002/05/13 by abeaudin@abeaudin_r400_win_marlboro

added include path for tiling address calculations

Change 27500 on 2002/05/10 by csampayo@fl_csampayo2_r400

Adding VGT test

Change 27333 on 2002/05/10 by mkelly@fl_mkelly_r400_win_laptop

* Renamed vfd_sanity.cpp to r400vfd_sanity_01.cpp
* Checkpoint on vfd for hos in r400vfd_hos_01.cpp

Change 27215 on 2002/05/09 by ashishs@fl_ashishs_r400_win

2 tests done, still texture and shading functions to be added.

test # 1: r400cl_gband_04.cpp
Guard Band Clipping test.(right now just all triangles gouraud shaded and not texture mapped)
VERTEX_DATA - TRIANGLE_STRIP, Top Left Quadrant, gouraud shading
INDEXES - TRIANGLE_STRIP, Top Right Quadrant, flat shading 0
VERTEX_LIST - TRIANGLE_STRIP, Lower Right Quadrant, flat shading 1
VERTEX_DATA - TRIANGLE_STRIP, Lower Left Quadrant, six textures, flat shading 0
Method: 6 Vert Strip
Expected Results: 4 triangles, four quadrants

test # 2: r400cl_clip_space_dx_ogl_01.cpp
This test is to check the OGL v/s DX clip space definition.
The D3D space is checked with the Z positions. As soon as the
primitive enters the -ve z positions in the D3D space the
primitive gets clipped but doesnt get clipped in the OpenGL space.
This is shown using 4 triangles. Left top and bottom correspond to D3D space whereas the right top and bottom traingles which are not clipped correspond to OpenGL space.

Change 27161 on 2002/05/09 by csampayo@fl_csampayo_r400

Updates

Change 27017 on 2002/05/08 by mkelly@fl_mkelly_r400_win_laptop

Update test description, will need to update again for subpixel mask and new state control for sampling.

Change 26999 on 2002/05/08 by mkelly@fl_mkelly_r400_win_laptop

Enhance VFD to support vertex point size in the shader import and export. See test r400vte_simple_point_01.cpp

Change 26932 on 2002/05/08 by csampayo@fl_csampayo2_r400

Updates

Change 26845 on 2002/05/07 by csampayo@fl_csampayo2_r400

Updates

Change 26838 on 2002/05/07 by mkelly@fl_mkelly_r400_win_laptop

Checkpoint on MSAA test which validates URC, LLC, and LRC offsets from LLC subsample points. 90% complete.

Change 26727 on 2002/05/07 by smoss@smoss_crayola_win

SU tests

Change 26707 on 2002/05/07 by ashishs@fl_ashishs_r400_win

144 sets of gourand-shaded clipped lines are drawn. all combinations of frustum clip codes w/ vertices 27^3 have been tested.

Change 26700 on 2002/05/07 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 26568 on 2002/05/06 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 26567 on 2002/05/06 by smoss@smoss_crayola_win

Update of Golds

Change 26550 on 2002/05/06 by smoss@smoss_crayola_win

Missing some files for these tests

Change 26461 on 2002/05/06 by ashishs@fl_ashishs_r400_win

changed : unpacking of colors added

Change 26391 on 2002/05/03 by mkelly@fl_mkelly_r400_win_laptop

Multi Sample Anti-Aliasing, 8 subsample, single subsample test on ULC.

Change 26369 on 2002/05/03 by smoss@smoss_crayola_win

su makefile

Change 26368 on 2002/05/03 by smoss@smoss_crayola_win

Golds

Change 26365 on 2002/05/03 by llefebvr@llefebvre_laptop_r400_emu

This is the new control flow sequencer. Expect things to be a bit unstable while this major change settles in. I know I broke 1 regression test (r400vgt_index_size_01) but the integration took so long that I decided to check the change in anyways and fix the problem from the TOTT. Sorry for the inconvenience.

Change 26223 on 2002/05/03 by ashishs@fl_ashishs_r400_win

Description:Legacy UCP clip test converted to R400 PrimLib,Former test name: tcl_ucp_combos_00 (R100,R200,R300)
Total of 720 triangles. Each triangle is stepped on the Y axis from -359.5 to +359.5
X values remain constant for all 720 triangles.
W = 360
All UCPS are vertically defined.
UCP definitions intersect each triangle at varying X locations.
UCP locations are cycled through 720 combinations of order on the X axis which is 6! (6*5*4*3*2*1).
Each of the 64 ucp_combos tests has an incrementally different triangle X location from -217 to +224, in 7 integer unit steps with UCPs adjusted accordingly.
Clipping enabled, all UCPs enabled
VTE performs 1/W reciprocal, X * 1/W, Y * 1/W, Z * 1/W
VTE performs offset and scale of 360
Guard Band Clipping is at 1.0f for vertical and horizontal adjust/discard.

Expected Results:720 small, clipped triangles from y = -360 to y = +360 (they appear towards middle of viewport along an imaginary edge and jut out to right); colors range from green-blue on the left hand side of the triangle "bar" to bright orange

Change 26138 on 2002/05/02 by smoss@smoss_crayola_win

SU tests and golds

Change 26103 on 2002/05/02 by smoss@smoss_crayola_win

SU tests, golds

Change 26044 on 2002/05/02 by ashishs@fl_ashishs_r400_win

nothing has been changed

Change 25986 on 2002/05/02 by mkelly@fl_mkelly_r400_win_laptop

Update test...

Change 25979 on 2002/05/02 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 25916 on 2002/05/01 by csampayo@fl_csampayo_r400

Updated Makefiles for added VGT test

Change 25910 on 2002/05/01 by csampayo@fl_csampayo_r400

Adding golds for vgt test

Change 25894 on 2002/05/01 by mkelly@fl_mkelly_r400_win_laptop

Jittered Super Sampling 2x1 2 pixel

Change 25818 on 2002/05/01 by ashishs@fl_ashishs_r400_win

nothing has been changed

Change 25775 on 2002/05/01 by mkelly@fl_mkelly_r400_win_laptop

Update...

Change 25576 on 2002/04/30 by ashishs@fl_ashishs_r400_win

180 gouraud shaded, clipped triangles.Vertex position combinations tested against frustum clipping.

Change 25569 on 2002/04/30 by ashishs@fl_ashishs_r400_win

180 gouraud shaded, clipped triangles.Vertex position combinations tested against frustum clipping.

Change 25557 on 2002/04/30 by ashishs@fl_ashishs_r400_win

144 gouraud shaded, clipped triangles.Vertex position combinations tested against frustum clipping.

Change 25539 on 2002/04/30 by ashishs@fl_ashishs_r400_win

255 gouraud shaded, clipped triangles.Vertex position combinations tested against frustum clipping.

Change 25514 on 2002/04/30 by ashishs@fl_ashishs_r400_win

change made due to upgrade in emulator

Change 25454 on 2002/04/29 by csampayo@fl_csampayo_r400

Adding VGT index size test

Change 25417 on 2002/04/29 by mkelly@fl_mkelly_r400_win_laptop

2 pixel jss, short test which works and 1 test which produces the below output:

ERROR: base class tcl_fifo write when full condition
Assertion failed, ../../../../emu_lib/model/gfx/pa/pasu/tcl_fifo.cpp@53: 0

Change 25196 on 2002/04/26 by csampayo@fl_csampayo_r400

Adding VGT index offset tests

Change 25186 on 2002/04/26 by mkelly@fl_mkelly_r400_win_laptop

Jitter Super Sample Test, currently times out and waiting on debug from Clay.

Change 25140 on 2002/04/26 by ashishs@fl_ashishs_r400_win

r400cl_frustrum_03 test, Converted from R200. 180 gouraud shaded, clipped triangles.

Change 25029 on 2002/04/25 by ashishs@fl_ashishs_r400_win

test file to see the working of perforce

Change 25027 on 2002/04/25 by ashishs@fl_ashishs_r400_win

test file to see the working of perforce

Change 25026 on 2002/04/25 by csampayo@fl_csampayo_r400

Updated for proper TRIANGLE_WITH_WFLAGS handling

Change 25000 on 2002/04/25 by ashishs@fl_ashishs_r400_win

Test r400 frustrum clip test, converted from r200

Change 24934 on 2002/04/25 by csampayo@fl_csampayo_r400

Adding VGT index min/max clamping tests

Change 24896 on 2002/04/25 by mkelly@fl_mkelly_r400_win_laptop

Update gold image, color changed by 1 bit in the blue channel for some reason...

Change 24894 on 2002/04/25 by mkelly@fl_mkelly_r400_win_laptop

Remove const from INDEX_BUFFER instance...

Change 24806 on 2002/04/24 by mkelly@fl_mkelly_r400_win_laptop

Jitter testing in progress....

Change 24712 on 2002/04/24 by smoss@smoss_crayola_win

updated su tests

Change 24645 on 2002/04/23 by csampayo@fl_csampayo_r400

Adding VGT reuse depth tests

Change 24556 on 2002/04/23 by mkelly@fl_mkelly_r400_win_laptop

Check point, test authoring in progress....

Change 24092 on 2002/04/19 by csampayo@fl_csampayo_r400

Various VGT tests. Note: most of these test have problems running due to emulator problems

Change 23955 on 2002/04/19 by smoss@smoss_crayola_win

su test for culling

Change 23919 on 2002/04/19 by mkelly@fl_mkelly_r400_win_laptop

Special triangle clip test where W=0

Vtx 0 = -0.50, -0.25, 0.00, 1.00
Vtx 1 = 0.25, -0.50, 0.00, 1.00
Vtx 2 = 0.50, 0.50, 0.00, 0.00

VTE out = ((1/W)*X)+X_OFFSET, ((1/W)*Y)+Y_OFFSET, ((1/W)*Z)+Z_OFFSET

Change 23831 on 2002/04/18 by mkelly@fl_mkelly_r400_win_laptop

update

Change 23828 on 2002/04/18 by mkelly@fl_mkelly_r400_win_laptop

Legacy Clipping UCP combos test converted to R400.

Change 23711 on 2002/04/17 by johnchen@johnchen_r400_win_marlboro

fix precision probelm for color and depth

Change 23521 on 2002/04/16 by smoss@smoss_crayola_win

setup unit tests

Change 23474 on 2002/04/16 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 23333 on 2002/04/15 by mkelly@fl_mkelly_r400_win_laptop

Converted legacy frustum clip test to demonstrate conversion methodology of an R200 clip test to R400.

Change 23165 on 2002/04/12 by csampayo@fl_csampayo_r400

Added VGT tests...

Change 23122 on 2002/04/12 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 23091 on 2002/04/12 by csampayo@fl_csampayo_r400

VGT test stressing primitive type vs index source

Change 23069 on 2002/04/12 by mkelly@fl_mkelly_r400_win_laptop

Update to handle when compare list is empty, so only output is generated.

Change 23036 on 2002/04/12 by mkelly@fl_mkelly_r400_win_laptop

Update

Change 23033 on 2002/04/12 by mkelly@fl_mkelly_r400_win_laptop

Simple VTE check for scale and offset

Change 22997 on 2002/04/12 by mkelly@fl_mkelly_r400_win_laptop

Update total PA validation test list...

Change 22996 on 2002/04/12 by mkelly@fl_mkelly_r400_win_laptop

Update W2K registry settings for dump enables...

Change 22995 on 2002/04/12 by mkelly@fl_mkelly_r400_win_laptop

Simple window offset test....

Change 22916 on 2002/04/11 by csampayo@fl_csampayo_r400

Added Makefile and SU parallelogram orientation tests

Change 22892 on 2002/04/11 by mkelly@fl_mkelly_r400_win_laptop

Simple clip test...

Change 22872 on 2002/04/11 by mkelly@fl_mkelly_r400_win_laptop

Basic clip rectangle checks...

Change 22861 on 2002/04/11 by mkelly@fl_mkelly_r400_win_laptop

Clip rectangle 0 control, 64 permutations...

Change 22813 on 2002/04/11 by mkelly@fl_mkelly_r400_win_laptop

72 triangle mesh, 1 packet, scissor test...

Change 22609 on 2002/04/10 by mkelly@fl_mkelly_r400_win_laptop

Update test to use the PrimLib DRAW_COMMAND_LOAD class, but still waiting on RBIU back pressure valve for this test to complete.

Change 22510 on 2002/04/09 by mkelly@fl_mkelly_r400_win_laptop

Scissor rect tests...

Change 22317 on 2002/04/08 by mkelly@fl_mkelly_r400_win_laptop

Simple scissor rectangle test...

Change 22293 on 2002/04/08 by mkelly@fl_mkelly_r400_win_laptop

SC scissor rectangle test

Change 22156 on 2002/04/05 by mkelly@fl_mkelly_r400_win_laptop

Clip enabled, no clipping, test currently fails...

Change 22155 on 2002/04/05 by mkelly@fl_mkelly_r400_win_laptop

Renaming to assure unique test names in test_lib...

Change 22145 on 2002/04/05 by mkelly@fl_mkelly_r400_win_laptop

TOP edge fill rule testing...

Change 22045 on 2002/04/05 by csampayo@fl_csampayo_r400

New VGT tests

Change 22024 on 2002/04/05 by mkelly@fl_mkelly_r400_win_laptop

Update tests to allocate additional memory during run...

Change 21761 on 2002/04/03 by mkelly@fl_mkelly_r400_win_laptop

Multi-packets testing raster filling...

Change 21758 on 2002/04/03 by mkelly@fl_mkelly_r400_win_laptop

Test case to demonstrate potential bug...

Change 21658 on 2002/04/03 by mkelly@fl_mkelly_r400_win_laptop

Stamp the output log file with the sync that the test was run against.

Change 21655 on 2002/04/03 by mkelly@fl_mkelly_r400_win_laptop

SC tests...

Change 21632 on 2002/04/03 by mkelly@fl_mkelly_r400_win_laptop

Generate 625 packets, 1250 triangles - checking basic SC functionality...

Change 21608 on 2002/04/03 by mkelly@fl_mkelly_r400_win_laptop

"regress_r400"

- include sync # on output regression directory
- automatically find the root directory on the client and send output to a directory called "$root/regress_r400"

This script "regress_r400" is intended for use by the validation team only.

All emu and PrimLib code writers are to continue using "make regress_e" at the test_lib/src/chip level for SANITY before checking in code.

Change 21535 on 2002/04/03 by mkelly@fl_mkelly_r400_win_laptop

W2K registry settings for debug dump control...

Change 21417 on 2002/04/02 by mkelly@fl_mkelly_r400_win_laptop

Update description...

Change 21379 on 2002/04/02 by mkelly@fl_mkelly_r400_win_laptop

Reference Bugzilla Bug 225...

Change 21376 on 2002/04/02 by mkelly@fl_mkelly_r400_win_laptop

Update support in regress_r400 for several more dumps including VGT, and reciprocal...

Change 21357 on 2002/04/02 by mkelly@fl_mkelly_r400_win_laptop

Update PA regress_r400 script to include comparison support for pm4RBDump.txt file whis is generated when W2K registery keyword pm4RBDump is true. This dump is helpful to check if PrimLib data is valid before CP processing.

Change 21250 on 2002/04/01 by mkelly@fl_mkelly_r400_win_laptop

Reference Bugzilla, bug 223...

Change 21065 on 2002/03/29 by mkelly@fl_mkelly_r400_win_laptop

Sync to this and UNIX will work....

Change 20960 on 2002/03/28 by mkelly@fl_mkelly_r400_win_laptop

Enabling 2 PrimLib tests in regress_e, removed user_chip_interface includes which are no longer required.

Change 20858 on 2002/03/28 by mkelly@fl_mkelly_r400_win_laptop

Add SC fill test for regress_e...

Change 20820 on 2002/03/28 by mkelly@fl_mkelly_r400_win_laptop

Change pitch in test...

Change 20816 on 2002/03/28 by mkelly@fl_mkelly_r400_win_laptop

Update tests due to change in Primlib setting...

Change 20507 on 2002/03/26 by mkelly@fl_mkelly_r400_win_laptop

Fixed the PA regression script to work with the newly relocated validation tree structure. Renamed the script.

Change 20506 on 2002/03/26 by mkelly@fl_mkelly_r400_win_laptop

Fixed all of the validation make files. The Depth value must be modified to match the directory depth from test_lib.

Change 20479 on 2002/03/26 by abeaudin@abeaudin_r400_win_marlboro

more moving of test files

Change 20475 on 2002/03/26 by abeaudin@abeaudin_r400_win_marlboro

move test_lib/src/sys to test_lib/src/chip/sys
move test_lib/src/gfx to test_lib/src/chip/gfx

Change 20473 on 2002/03/26 by abeaudin@abeaudin_r400_win_marlboro

moving test directories round

Change 12489 on 2002/01/07 by rbeaudin@rbeaudin_r400_win_marlboro

removed unused tests in sanity

Change 11682 on 2001/12/13 by rbeaudin@rbeaudin_r400_win_marlboro

removed ray test

Change 9709 on 2001/11/12 by sallen@sallen_r400_unix_marlboro

new ferret_emu.h
pipeline points to new place (not final place, though)

Change 9315 on 2001/11/06 by kmahler@kmahler

Changes to support Shader program testcase, "onetri_vshader.cpp".

Change 9080 on 2001/11/02 by sallen@devel_sallen

make sure "NO_FERRET" switch removes all ferret
(so I can track down namespace problems....)

Change 9039 on 2001/11/01 by rbeaudin@ma_rayb1

fixed unix segment error

Change 8839 on 2001/10/30 by sallen@devel_sallen

ferret updates
- add test_lib/ferret/ex1 example unit test
- update building for unit and block testing
- add memory pipe data structure to ferret (& comparitor) - though not hooked up yet

Change 8547 on 2001/10/26 by hwise@fl_hwise

Adding Ray's modified onetri_index_plgx test (under new name)

Change 8541 on 2001/10/26 by hwise@fl_hwise

Updated R400 with latest R300 emulator baseline.

Affected block files: BIF, CP, VIP

```
Misc:
  1) Updated BIF, CP, MC, and VIP functionality with R300 mods
  2) Added pm4capture library (used for dumping pm4 streams for debug)
  3) Renamed the sanity test onetri_indexes_plgx.cpp to
     onetri_indexes_plgx_ray.cpp since it contains block level testing
     stuff.
  4) Removed block level testing stuff from onetri_indexes_plgx.cpp
  5) Updated plgx pm4 functions to match updated CP register spec
     and added some bug fixes done for R300
```

Change 8280 on 2001/10/23 by rbeaudin@MA_RAYB

added new block interfaces

Change 7744 on 2001/10/15 by rbeaudin@MA_RAYB

new test

Change 7700 on 2001/10/12 by sallen@devel_sallen

make compile on NT & Unix

Change 7686 on 2001/10/12 by sallen@ma_sallen

add ferret targets for windows

Change 7561 on 2001/10/10 by sallen@devel_sallen

merge all the ferret code in

Change 7528 on 2001/10/10 by sallen@devel_sallen

merge in first round of ferret changes

Change 6719 on 2001/09/20 by rbeaudin@MA_RAYB

changing location of interfaces

Change 6714 on 2001/09/19 by rbeaudin@MA_RAYB

emu compiles but test program does not

Change 6638 on 2001/09/18 by rbeaudin@MA_RAYB

more crayola name changes

Change 6308 on 2001/09/11 by rbeaudin@MA_RAYB

change to khan

Change 5829 on 2001/08/28 by tyroneh@devel_tyroneh

<corrected a makefile syntax error that would result in an error on the UNIX side, but pass on the windows side.
-lmodel, instead of "- lmodel">

Change 5827 on 2001/08/28 by tyroneh@devel_tyroneh

<added a search path for the model directory>

Change 5560 on 2001/08/21 by rbeaudin@MA_RAYB

old working program

Change 5331 on 2001/08/14 by rbeaudin@MA_RAYB

test no gfx

Change 5164 on 2001/08/09 by rbeaudin@MA_RAYB

block A works

Change 5067 on 2001/08/07 by rbeaudin@MA_RAYB

interface stuff

Change 4670 on 2001/07/24 by hwise@fl_hwise2

Write to new register

Change 4664 on 2001/07/24 by hwise@fl_hwise2

Add basic test

ATI TECHNOLOGIES INC.

R400 Program

CONFIDENTIAL

ATI TECHNOLOGIES INC.

December 13th, 2001

R400 Program Review

December 13th, 2001

Peter Pellerite

CONFIDENTIAL

Agenda

Welcome	Peter Pellerite	10:00 am
Marketing (Features)	Ray Thompson	10:10 am
Architecture	Andy Gruber	10:25 am
Logic Design		
Toronto Design Team	Lili Sinclair	10:50 am
Orlando Design Team	Joe Cox	11:10 am
Marlboro Design Team	Mark Fowler	11:40 am
Lunch Break		***~12:30 pm***

Agenda

Performance Estimation	Steve Morein	1:10 pm
Software	Kerry Wilkinson	1:40 pm
Emulation / Simulation Env.	Paul Mitchell	2:00 pm
Libraries / Technology	Raj Verma	2:30 pm
Front-End to Back-End Flow	Mark Sprague	3:00 pm
Break		***~3:30 pm***
DFT	D. Hsiung/R.Treuer	3:40 pm
IC Packaging	Vincent Chan	3:55 pm
Program Issues Discussion	All	4:10 pm
Wrap-up	Peter Pellerite	4:45 pm

CONFIDENTIAL

R400 Organization

R400 Team

Peter Pellerite
R400 Program Manager

Ray Thompson
Marketing Manager

Andy Gruber
Architectural Manager

Steve Morein
R400 Architectural Lead

Clay Taylor
Orlando Architectural Manager

David Glen
Toronto Architectural Lead

Mark Fowler
Marlboro Hardware Manager

Mike Mantor
Orlando Hardware Manager

Lili Sinclair
Toronto Hardware Manager

Paul Mitchell
Test Environment Manager

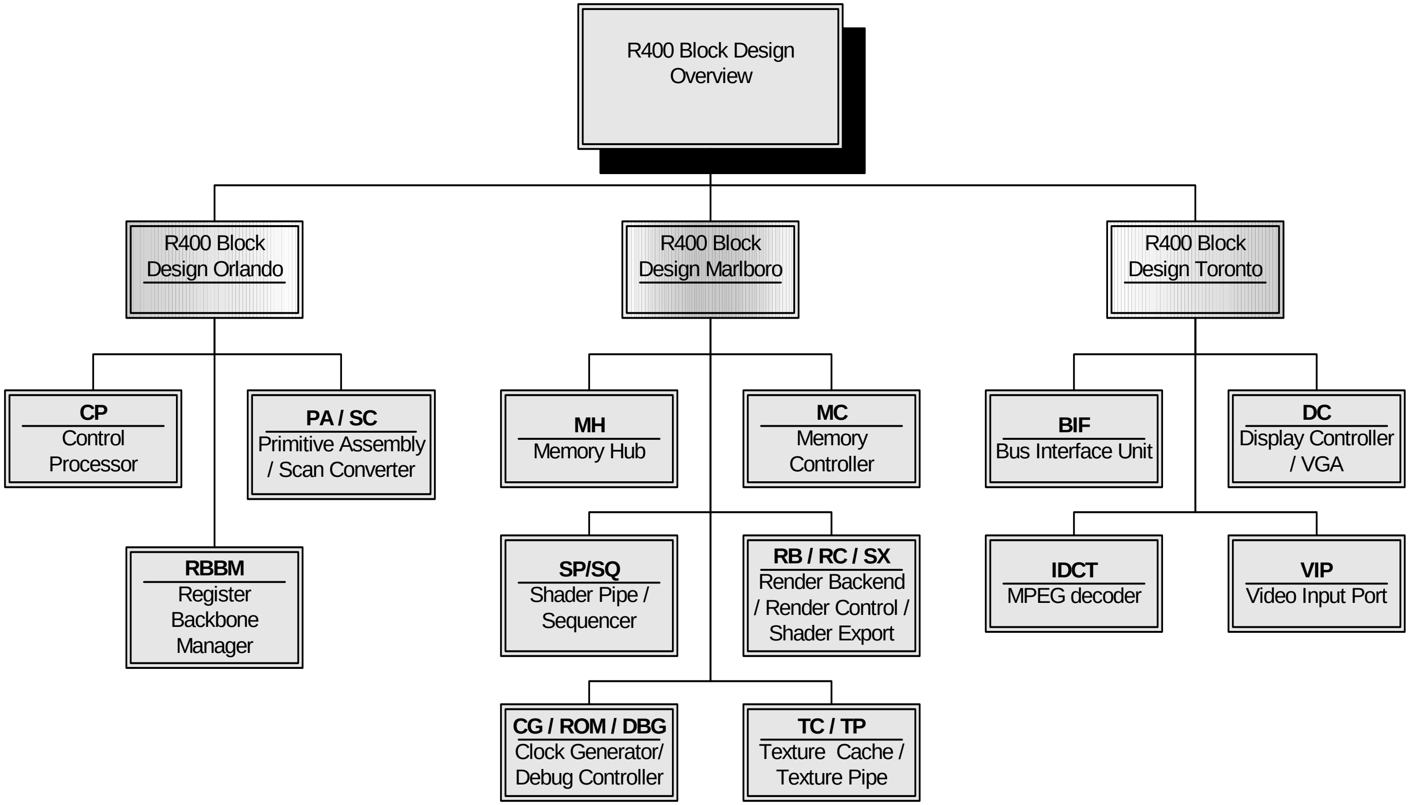
Christeen Gray
Orlando System Validation and Test Environment Manager

Kerry Wilkinson
Software Manager

Phil Rogers
Software Architect

CONFIDENTIAL

Block Responsibility


R400 Block Design Overview
R400 Block Design Orlando
R400 Block Design Marlboro
R400 Block Design Toronto
CP
Control Processor
PA / SC
Primitive Assembly / Scan Converter
RBBM
Register Backbone Manager
MH
Memory Hub
MC
Memory Controller
SP/SQ
Shader Pipe / Sequencer
RB / RC / SX
Render Backend / Render Control / Shader Export
CG / ROM / DBG
Clock Generator/ Debug Controller
TC / TP
Texture Cache / Texture Pipe
BIF
Bus Interface Unit
DC
Display Controller / VGA
IDCT
MPEG decoder
VIP
Video Input Port

CONFIDENTIAL

ATI TECHNOLOGIES INC.

December 13th, 2001

ATI TECHNOLOGIES INC.

R400 Program

CONFIDENTIAL

ATI TECHNOLOGIES INC. December 13th, 2001

Architecture

Prepared by: Andrew Gruber

CONFIDENTIAL

ATI TECHNOLOGIES INC.

December 13th, 2001

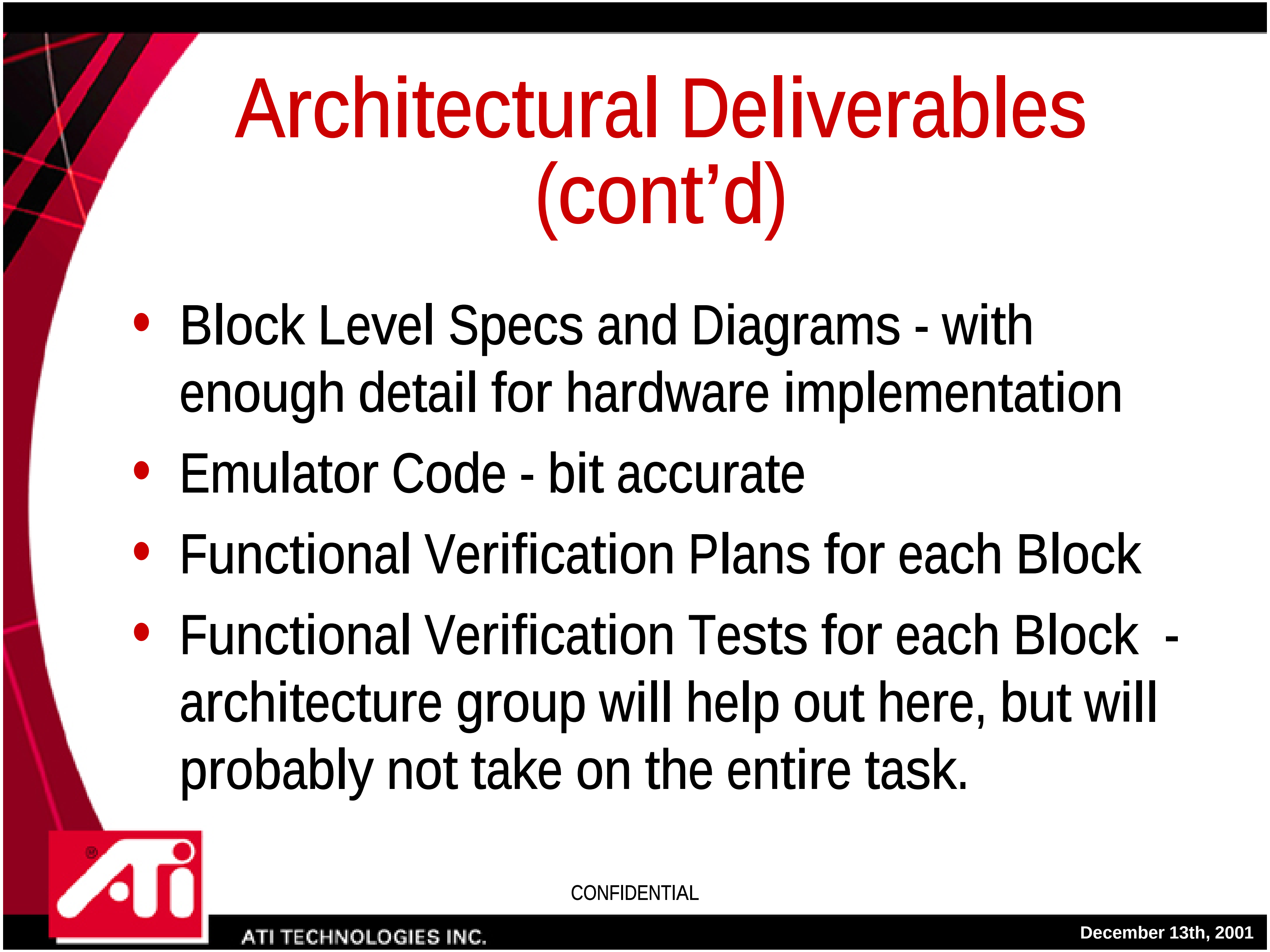
Topics

- Plans / Deliverables
- Current Status
- Open Issues / Concerns
- Schedule

CONFIDENTIAL

Architecture Deliverables

- Top Level Documentation
  - Block Diagram
  - Suggested Floor Plan
  - Feature List (in conjunction with Marketing)
  - Area Estimate (in conjunction with Block Leads)
  - Programming Guide
  - Shader Guide
  - Detailed Register Specification
  - State Management Documentation
  - Synchronization and Coherency Documentation
  - 2D Guide
  - Real Time and Multi-Media Guide
  - HOS Guide
  - Performance Verification Plan



Architectural Deliverables (cont'd)

- Block Level Specs and Diagrams - with enough detail for hardware implementation
- Emulator Code - bit accurate
- Functional Verification Plans for each Block
- Functional Verification Tests for each Block - architecture group will help out here, but will probably not take on the entire task.

CONFIDENTIAL

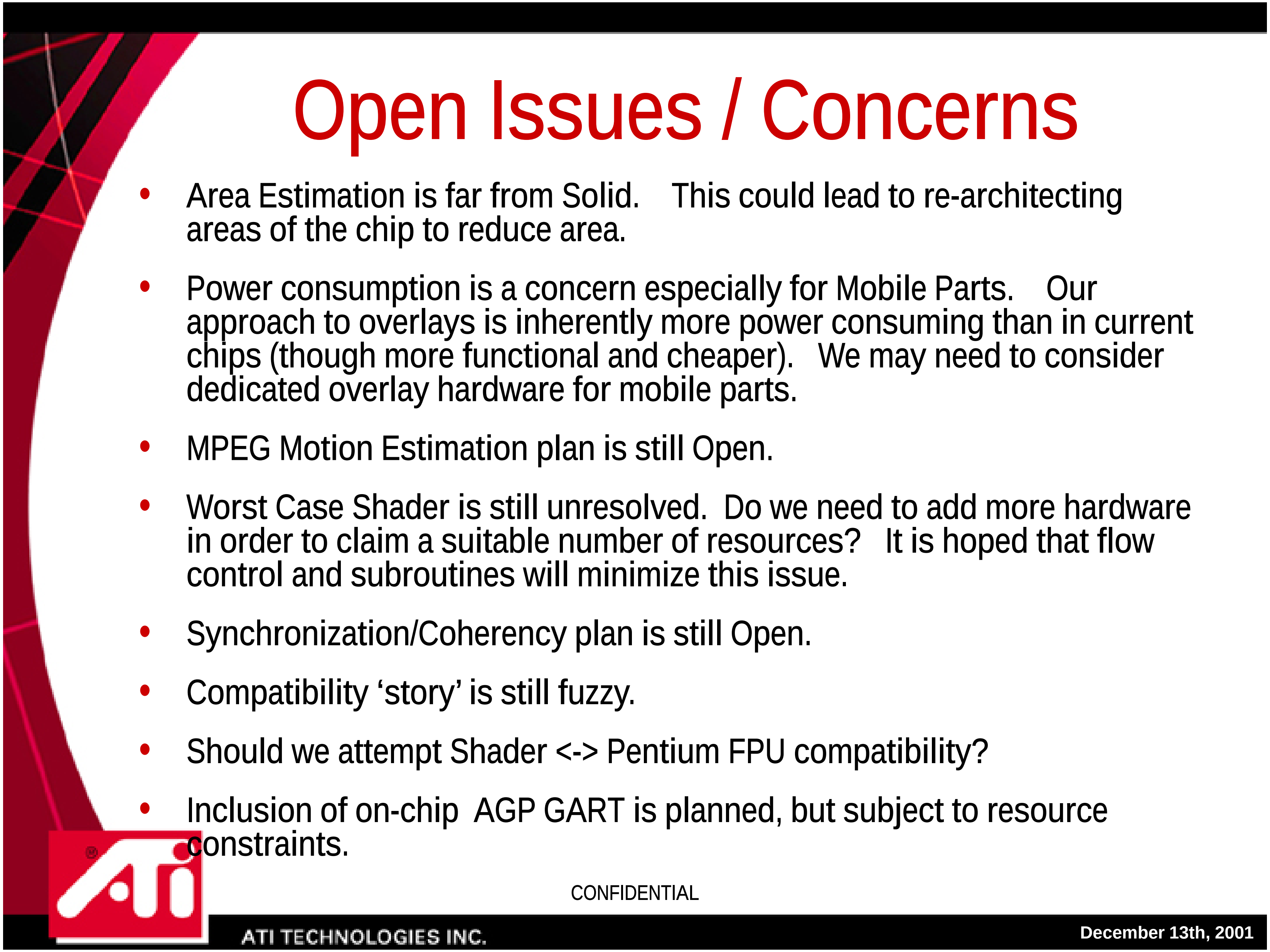
Current Status

- Feature Set nearly frozen - MPEG Motion Estimation still outstanding
- Top Level Architecture Resolved
- First Pass Register Spec Done
- First Pass Instruction Set Done
- Most Block-Level Specs Written and First Pass Reviews done
- Some Blocks have initial emulator code released
- Basic 2D implementation Resolved

CONFIDENTIAL

ATI TECHNOLOGIES INC.

December 13th, 2001



Open Issues / Concerns

- Area Estimation is far from Solid. This could lead to re-architecting areas of the chip to reduce area.
- Power consumption is a concern especially for Mobile Parts. Our approach to overlays is inherently more power consuming than in current chips (though more functional and cheaper). We may need to consider dedicated overlay hardware for mobile parts.
- MPEG Motion Estimation plan is still Open.
- Worst Case Shader is still unresolved. Do we need to add more hardware in order to claim a suitable number of resources? It is hoped that flow control and subroutines will minimize this issue.
- Synchronization/Coherency plan is still Open.
- Compatibility 'story' is still fuzzy.
- Should we attempt Shader <-> Pentium FPU compatibility?
- Inclusion of on-chip AGP GART is planned, but subject to resource constraints.

CONFIDENTIAL

ATI TECHNOLOGIES INC.

December 13th, 2001

Target Schedule

Milestone	Plan	Actual	Forecast
Top Level			
MPEG Plan	12/17		
Block Diagram	Done		
Suggested Floor Plan	Done		
2D Programming guide	12/19		
Shader Guide	12/19		
Programming Guide	12/24		
Area Estimate	2/1		
Synchronization and Coherency Document	1/15		
HOS Document	1/15		
Multi Media Guide	2/1		
Performance Verification Plan	2/1		

CONFIDENTIAL

Target Schedule

Milestone	Plan	Actual	Forecast
Block Level Specs complete	2/15		
Block Level Emulators Phase 1	1/15		
Block Level Emulators Phase 2	3/1		
Block Level Emulators Phase 3	4/15		
Block Verification Plans Complete	4/1		

ATI TECHNOLOGIES INC.

R400 Program Review

December 13, 2001

ATI TECHNOLOGIES INC. CONFIDENTIAL December 13th, 2001

ATI 2055
LG v. ATI
IPR2015-00326

SKILL ALLOCATION BY BLOCK

SKILL / UNIT	Requirements Ownership	Algorithm Development	Algorithm C-Sim / Emulator	Transfer to H/W Design	Design / Unit Level Validatoin	Valida-tion Test Plan	Test Gen / PrimLIb	Full Chip Env & Regress	IKOS / Diags	Phy. Design Process
Command Processor	Clay Taylor & Mike Mantor John Carey		Harry Wise	John Carey	Frank Liljeros Alex Ashkar Tushar Shah	J. Carey H. Wise	Carlos Sampayo Michael Kelly	Chris Gray Steve Moss	Michael Silver	Stevie Camlin
Vertex Grouper & Tesselator		Scott Hartog	Vineet Goel	Brian Buchner	Brian Buchner Scott Hartog Stevie Camlin	VG/SH Carlos Sampayo				
Clip / Viewport Xform	Mike Mang			Bob Hankinson		Mike Mang				
Setup	Mike Mang		Dan Clifton			Mike Mang				
Scan Converter			Randy Ramsey Mike Mantor		Randy Ramsey Donald Lee	RR/MM				

- HW Architecture Lead – Clay Taylor
- Hardware Design Lead – Mike Mantor
- System Validation & Test Environment – Christeen Gray

PRIMITIVE ASSEMBLY

- **CLIP / VIEWPORT XFORM**
- **SETUP ENGINE**
- **SCAN CONVERSION**
- **VERTEX GROUPER TESSELLATOR**

SETUP ENGINE

- Features
  - Supports Point/Line/Fill rendering modes.
  - Supports Polygon Offset
  - Back-Face Culling & Null-Prim Culling
  - Converts all Prims to 3-verts (Points->Rect, Lines->Parallelograms)
  - Perspective Correction Ena/Dis (On/Off for All attributes, no partial allowed).
  - Computes Min/Max Primitive Z for Hi-Z testing.
  - 1 Prim/Clk up to Back-Face, Null-Prim culling,
    2 Clks/Prim for I/W,J/W,1/W,Z Gradient calc's

- Issues
  - Determining design split for 2 Clks/Prim
  - Precision Analysis / Requirements for Z !!, I/W, J/W, 1/W

CLIP / VIEWPORT XFORM

Features

- Clipper similar to R100/200/300 except no Attribute Interpolation
  - (Significant design changes since R200/R300 vector engines were used for UCP)
- 6 User Clip Planes
- Clip and Discard Guard Band Support
- Point Clipping to UCP's
- Edge Flag support for Line/Point Fill Mode rendering.
- 1 Vtx/Clk (VTE similar to R200/R300)

Issues

- Work through RB Interfaces for post-vertex shader data.

SCAN CONVERTER

Features

- Rasterize Triangles, Rects (Points/Rects), Parallelograms (Lines)
- Supports 8k x 8k guard band with 4k x 4k rendering window
- Line Stipple Support (Not using textures)
- Multi-Directional Rect Walk for Overlapping Blit Support
- Export of Screen X,Y for use by Pixel Shader
- 1 Scissor Rect and 4 Clip Rectangles
- 1 thru 8 sample AA rasterization and centroid computat’n
- Supports Real-Time Stream Interrupt Rasterization
- Supports Visibility Query
- Supports Multi-Pass Pixel Shader Pixel Culling
- 1 8x8 Tile/Clock Coarse Rasterization
- 2 2x2 Quads/Clock (8 pixels/clk) Detail Rasterization w/ up to 8 subsamples/pix @ same rate

Features NOT Included

- NO support for perpendicular end caps (R200 did not support it)
- NO Poly Stipple Support (Will use Pixel Shader)

SCAN CONVERTER OPEN ISSUES

- 2D Lines – Legacy compliant
- Z Tile Corner Precision Analysis/Requirements
- BaryC Coeff Precision Analysis/Requirements
- AA subsample location determination (centroid calculations)
- Texture LOD fudge knowledge still needs to be transferred
- Real-Time Stream Interface Definition and Validation
- Multi-Pass Pix Shader still needs definition.
- Line Stipple Implementation needs definition and validation.

HOS FEATURES

- Continuous & Discrete Point-Normal Triangle
- Continuous & Discrete Point-Normal Quad
- Continuous & Discrete RT-Patches
- Includes Rational Patches
- Bezier, Bspline, Catmull-Rom surfaces
- Adaptive Tessellation of all of above surface primitives
- Flexible overall system allowing any type of tessellation
- Subdivision surfaces – Catmull-Clark, Loop
- Programmable Pre-HOS computations and vertex evaluation

VERTEX GROUPER TESSELLATOR

FEATURES

- NO support for traditional "vertex data in command stream" (Approved by SW)
- NO support for state-based vertices (part of the OGL-Friendly I/F) (Accepted by SW)
- Full DX/GL Prim type support
- Vertex Reuse across ~16 vertices (this likely will decrease with derivative chips)
- 2-D Draw Interface for 2D packets down 3D pipeline.
- Discrete / Continuous Tesselation parametric coordinate generation.
- Combining of Vertex Indices for Vtx Shader Primitive Operations
- Auto Index Generation for 2-pass optimized N-Patch Support (as well as Adaptive Tess of N-Patches).
- Supports 1 Vtx/Clk, 1 Prim/Clk whichever is slower (i.e. Tri List with no reuse would be 3 Clks/Prim since 3 unique vertices)
- Most HOS modes are 2 Clks/Vtx, but should match up ok with shader performance.

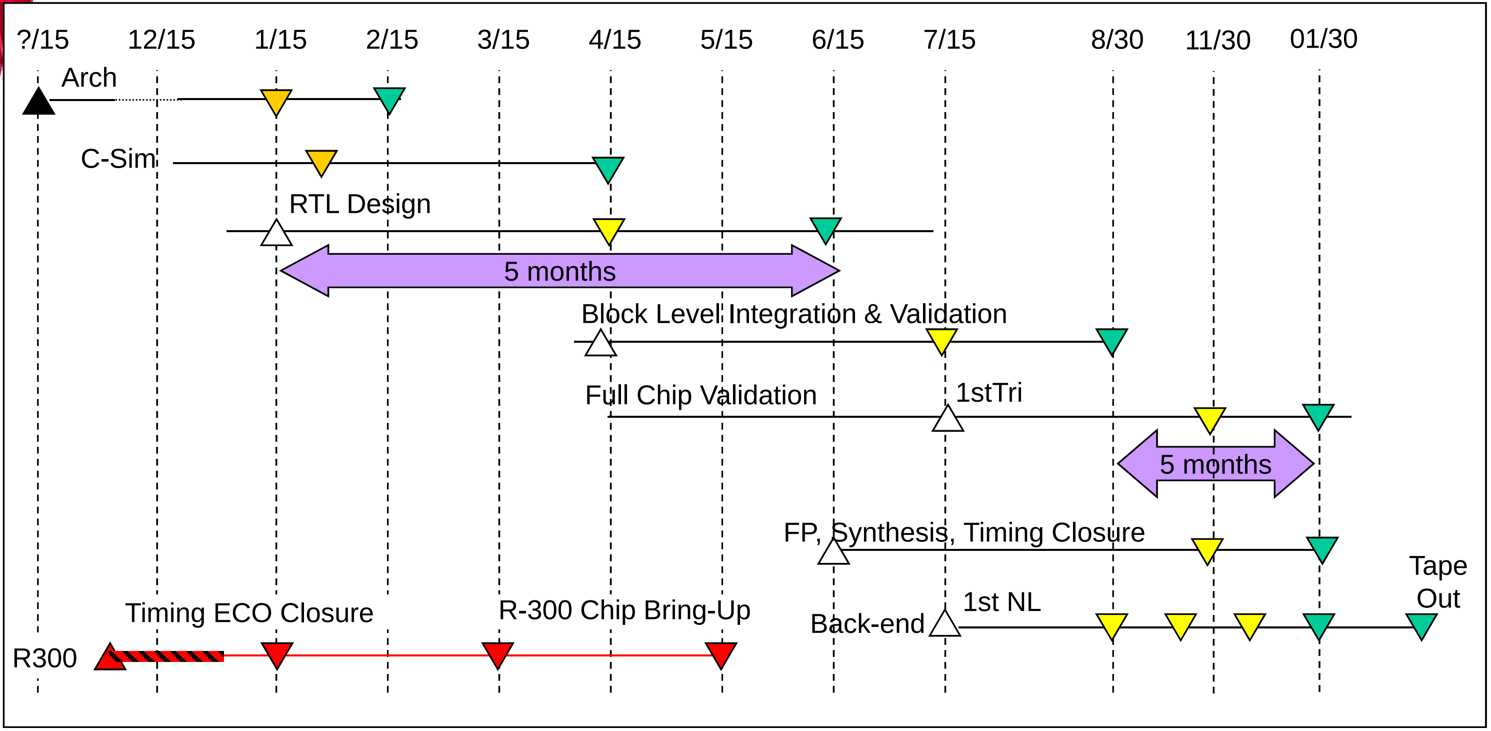
VGT "Issues"

- Packet grouping for 2D efficiency not fully understood across the 3D pipe.
- Competitive Risk with NO "Vtx Shader-Like Programmable" Tesselation Engine
- Microsoft…

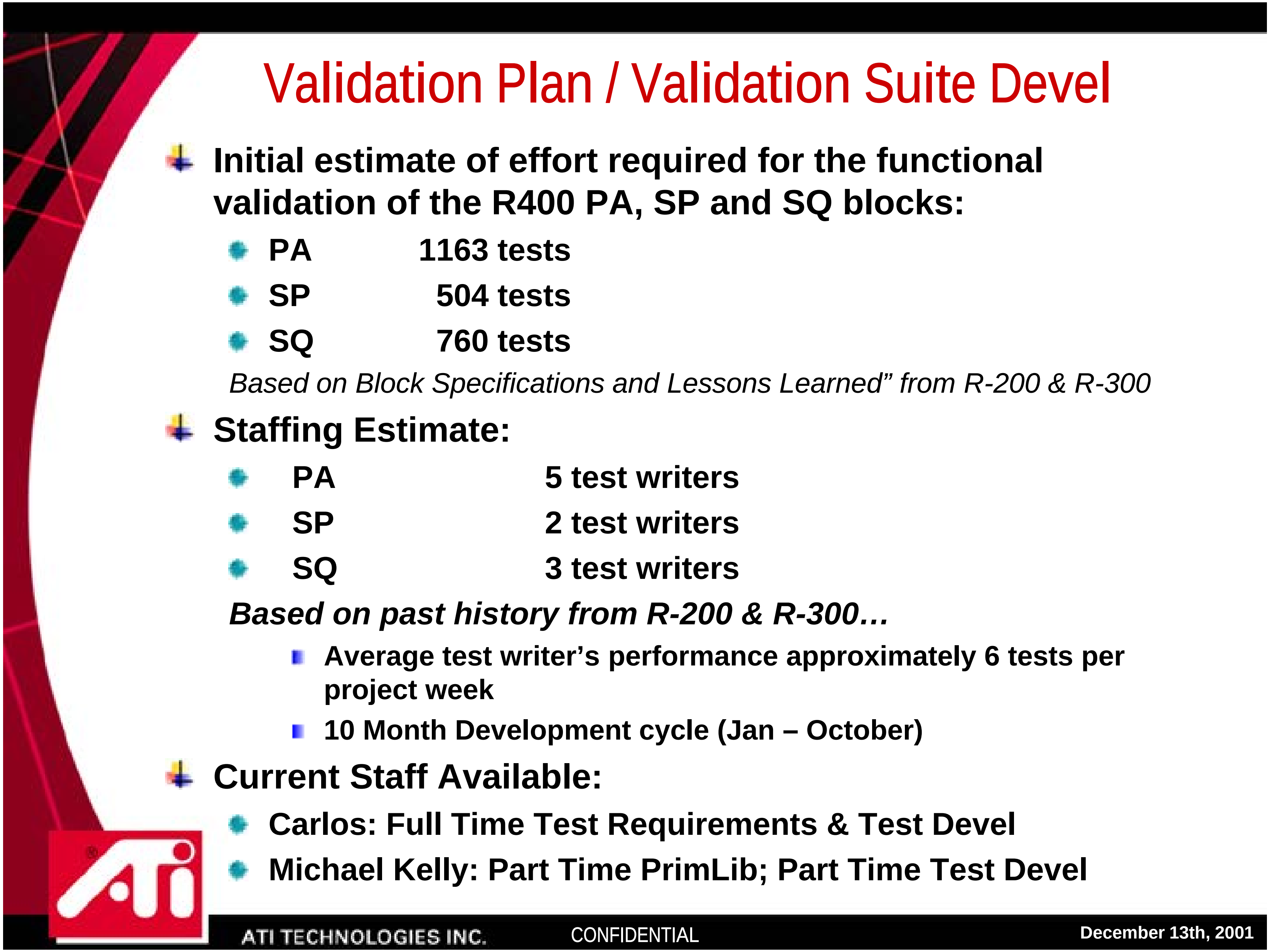
Command Processor / RBBM

- **Open Issues**

R-400 CP / RBBM / PA Schedule



- **Yellow Milestones 70-80% Complete**
- **Green Milestones represent >90% Complete**



Validation Plan / Validation Suite Devel

- Initial estimate of effort required for the functional validation of the R400 PA, SP and SQ blocks:

PA	1163 tests
SP	504 tests
SQ	760 tests

Based on Block Specifications and Lessons Learned" from R-200 & R-300

- Staffing Estimate:

PA	5 test writers
SP	2 test writers
SQ	3 test writers

Based on past history from R-200 & R-300…

  - Average test writer's performance approximately 6 tests per project week
  - 10 Month Development cycle (Jan – October)

- Current Staff Available:
  - Carlos: Full Time Test Requirements & Test Devel
  - Michael Kelly: Part Time PrimLib; Part Time Test Devel

ORLANDO RISKS & ISSUES

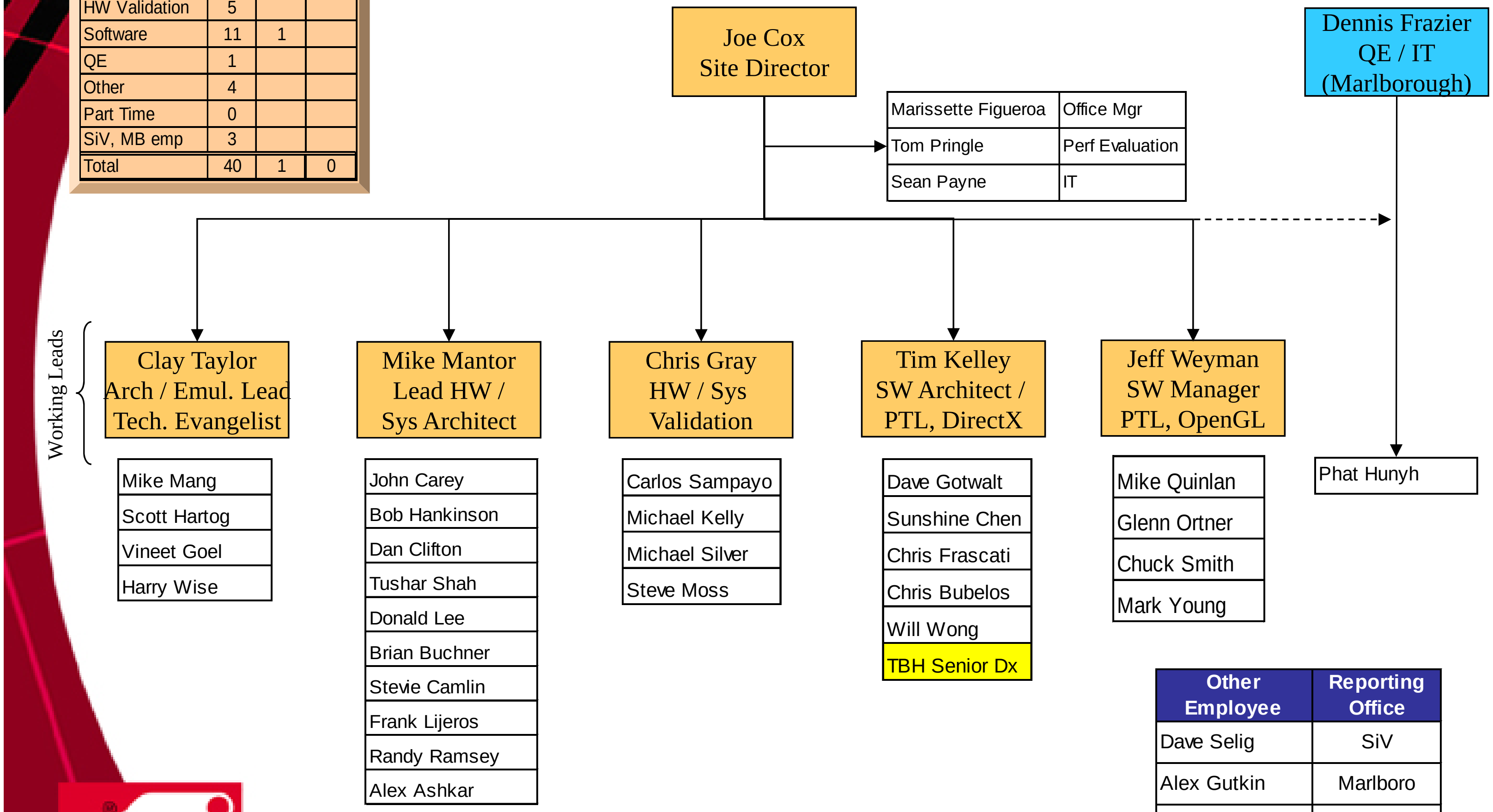
- R-300 Physical Design & Chip Bring-up
- IRIS Physical Design & Chip Bring-Up
- RV-350 / R-350 required support
- Major Chip-Wide Architectural Changes
  - Legacy Issues
  - Integration
  - Chip Validation
- Area Estimate – 1st cut January
- Validation / Diag Suite legacy
  - Staffing
  - “New” Test Environment
- Name: Crayola…

BACKUP SLIDES

ATI - Orlando Staffing

September 2001

Function	Filled	Req's Crit 1	Req's Crit 2
Arch, Sys Val'n	5		
Hardware	11		
HW Validation	5		
Software	11	1	
QE	1		
Other	4		
Part Time	0		
SiV, MB emp	3		
Total	40	1	0



Joe Cox
Site Director

Marissette Figueroa	Office Mgr
Tom Pringle	Perf Evaluation
Sean Payne	IT

Dennis Frazier
QE / IT
(Marlborough)

Clay Taylor
Arch / Emul. Lead
Tech. Evangelist

- Mike Mang
- Scott Hartog
- Vineet Goel
- Harry Wise

Mike Mantor
Lead HW /
Sys Architect

- John Carey
- Bob Hankinson
- Dan Clifton
- Tushar Shah
- Donald Lee
- Brian Buchner
- Stevie Camlin
- Frank Lijeros
- Randy Ramsey
- Alex Ashkar

Chris Gray
HW / Sys
Validation

- Carlos Sampayo
- Michael Kelly
- Michael Silver
- Steve Moss

Tim Kelley
SW Architect /
PTL, DirectX

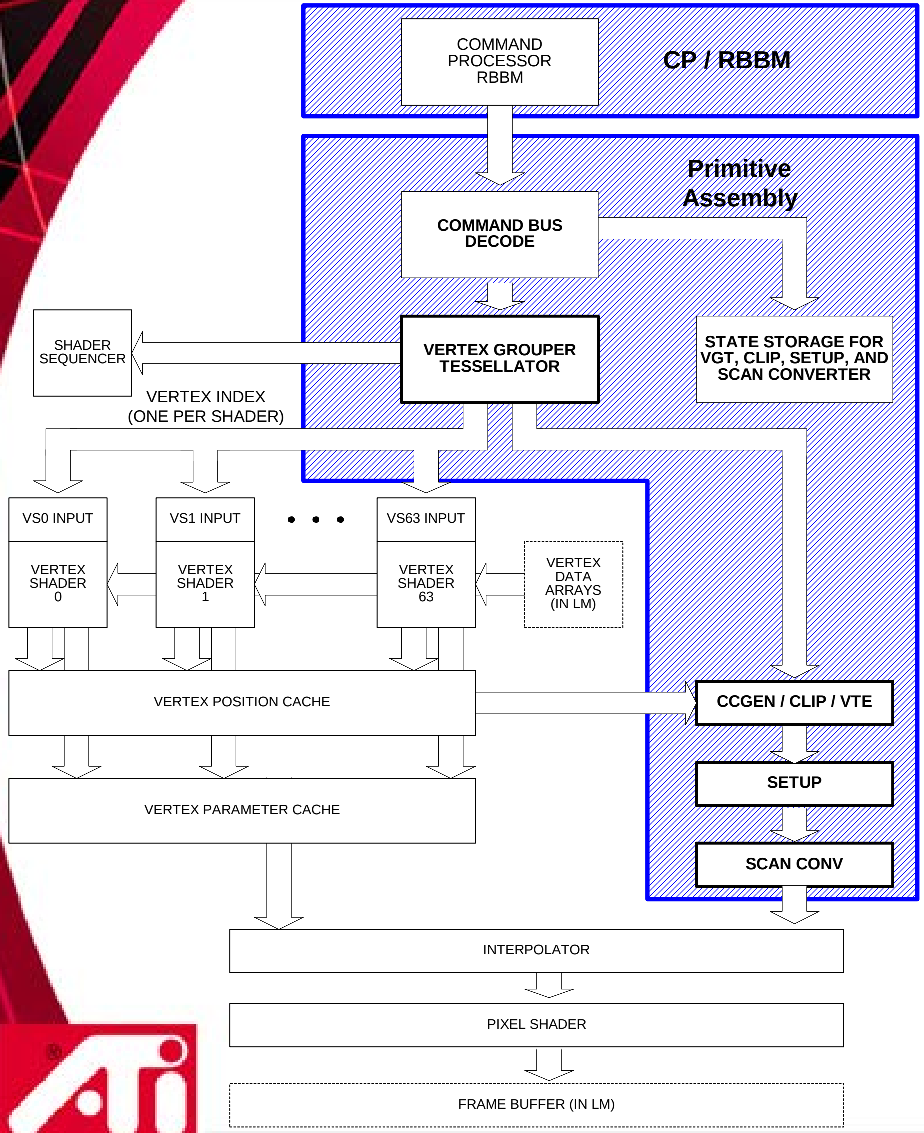
- Dave Gotwalt
- Sunshine Chen
- Chris Frascati
- Chris Bubelos
- Will Wong
- TBH Senior Dx

Jeff Weyman
SW Manager
PTL, OpenGL

- Mike Quinlan
- Glenn Ortner
- Chuck Smith
- Mark Young

Phat Hunyh

Other Employee	Reporting Office
Dave Selig	SiV
Alex Gutkin	Marlboro
Tony Delaurier	SiV



Top level Block Diagram – Orlando (to VGT centric)

- PRIMATIVE ASSEMBLY
  - VGT
  - CLIP / VTE
  - SETUP
  - SCAN CONVERTER
- COMMAND PROCESSOR
- RBBM

ATI TECHNOLOGIES INC.

R400 Program

CONFIDENTIAL

Marlboro Design

Prepared by: Mark Fowler

CONFIDENTIAL

ATI TECHNOLOGIES INC.

December 13th, 2001

Block Deliverables

- **SQ/SP – Shader Engine**
- **SX/RC/RB – Render Backend**
- **TP/TC – Texture Unit**
- **MH/MC – Memory Sub-System**
- **CG/ROM/DBG – Clock block, ROM controller, debug controller**
- **TST – Test Controller**

CONFIDENTIAL

Chip Deleiverables

- Graphics Controller (GC) Verification
- Chip Level Verification
- Chip Integration

CONFIDENTIAL

SQ/SP Team

- **SQ Arch / Emulation - Laurent Lefebvre**
- SP Arch / Emulation - **Andi Skende**
- **SQ HW Design - Vic Romaker**
  - <another person here>
- SP HW Design - Andi Skende
- SQ/SP Testbench / Regression
  - **Frank Hsien**

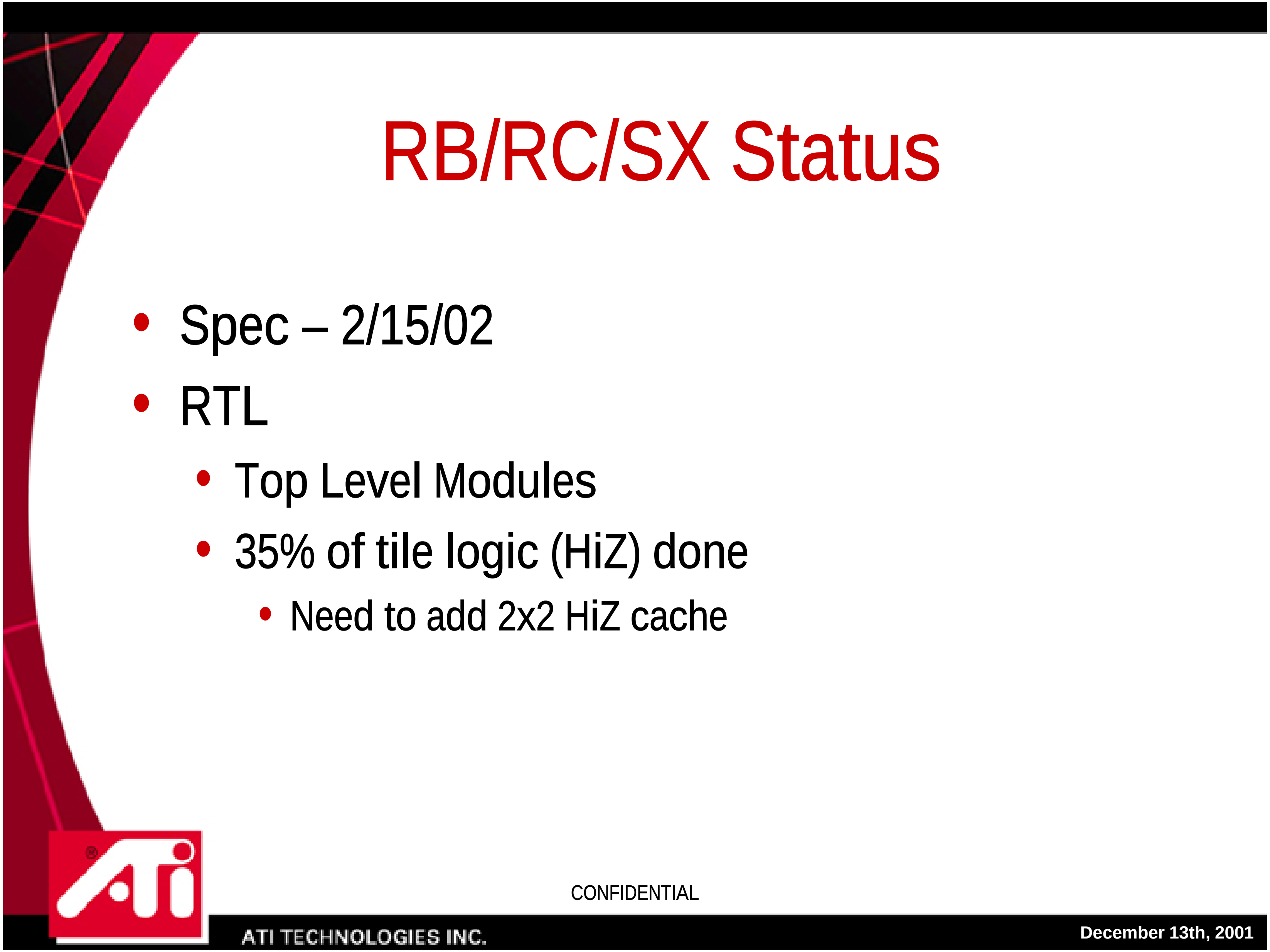
CONFIDENTIAL

SP/SQ Status

- SP spec – Ready for review < 1 week
- SP RTL
  - Top Level Module
  - Vector Unit – 70%
  - Scalar Unit – 40%
- SQ spec – 1/2/01
- SQ RTL – 5% (GPR allocation)

RB/RC/SX Team

- Architeture / Emulation – Larry Seiler
- HW Design Team
  - Jay Wilikinson
  - Bill Lawless
  - John Chen
- Testbench / Regressions
  - Yung Jiang



RB/RC/SX Status

- Spec – 2/15/02
- RTL
  - Top Level Modules
  - 35% of tile logic (HiZ) done
    - Need to add 2x2 HiZ cache

CONFIDENTIAL

ATI TECHNOLOGIES INC.

December 13th, 2001

MH Team

- **MH Architecture**
  - **Michael Doggett**
  - **Ken Correll**
- MH Hardware Design
  - Ken Correll
  - Paul Vella
- **MH/MC Testbench / Regressions**
  - **Ying Valcour**

CONFIDENTIAL

MC Team

- **MC Architecture**
  - **Larry Seiler**
  - **Ken Correll**
  - **Bob Bloemer**
- MC Hardware Design
  - Bob Bloemer
  - Bei Wang

MH/MC Status

- MH spec – 12/20/01
  - Some work on depth/stencil locks TBD
- MH RTL ~20% complete
- MC arch spec – 12/20/01
  - Closure on initialization / power
  - Number of DRAM types
  - Pad Interface
  - 500MHz DRAM
    - Read/write optimizations

TP / TC Team

- **TP Arch / Emulation - Jocelyn Houle**
- **TC Arch / Emulation - Michael Doggett**
- **TP/TC Deign Team**
  - **Tein Wei**
  - **Suba Durairajan**
  - **Ray Manapat**
- **TP/TC Testbench / Regressions**
  - **Steve Croce**

CONFIDENTIAL

TP / TC Status

- TC arch spec – 1/2/02
- TP arch spec – 1/2/02
- Minimal to no RTL entered and HW specification (Team on RV250)

CONFIDENTIAL

GC Verification

- Testbench include graphics engine, MH/MC, CP/RBBM
- Primary method of engine validation
  - 2D, HOS, AA, RTS, Shadows, MPEG, etc…
- Performance Validation (local memory only)
- Tests from block level tests and new ones w/ emphasis on distributed function
- Need resource (Verification Manager) to drive the plan, handle regressions, etc..

CONFIDENTIAL

Chip Verification

- Target for system level verification
  - DC <- MH/MC Validation
  - MPEG encode/decode w/ IDCT
  - Performance w/ AGP
  - MH/MC client arbitration
  - In general make sure blocks play nice
- **Need resource (Verification Manager) to drive the plan, handle regressions, etc..**
- Need to split effort among sites

CONFIDENTIAL

Chip Integration

- DFT – Plexus (Rick Fissette)
- CG, Power Management – Greg Sadowski
- Backend/Synthesis Flow – Mark Sprague, Rob Rouleau
- Diagnostics – Brian Leblanc
- I/O ring, timing constraints, pre and post layout timing, netlist delivery, feedthroughs, etc… - Rick Fuller, Jose Marsano, (plus 2 more)

Milestones

- 2/15/02
  - Programming specs (registers, shader guide, programming guide, etc..) complete and reviewed.
  - Block HW/Arch specs completed and reviewed
  - Feature Guides (2D, MPEG, Real Time, etc…) completed and reviewed
  - “Reasonable” area estimate using 0.13lv library for sample blocks (0.10 memory compiler)

CONFIDENTIAL

Milestones (cont)

- 4/15/02
  - Block RTL submitted to GC, enough functionality for gouraud shaded triangle
- 6/1/02
  - All blocks submitted to GC w/ 50% functionality (enough for basic perf testing)
  - Emulator complete
  - “Very reasonable” area estimate

CONFIDENTIAL

Milestones (cont)

- 8/1/02
  - All blocks submitted to GC w/ 80% functionality (enough for most perf testing)
- 9/15/02
  - All blocks submitted to GC, RTL feature complete
- 11/15/02
  - RTL freeze (debug complete)

CONFIDENTIAL

Milestones (cont)

- 1/15/03 - tapeout

CONFIDENTIAL

Issues / Risks / Concerns

- Other chips
  - RV250 tapeout, bringup
  - A5 engine
  - others?
- 400MHz (clock uncertainty still 300ps?)
- 500MHz Memory, Pad Interface
- # Features / Verification

ATI TECHNOLOGIES INC.

R400 Program

ATI 2057
LG v. ATI
IPR2015-00326

Agenda – January 17th, 2002

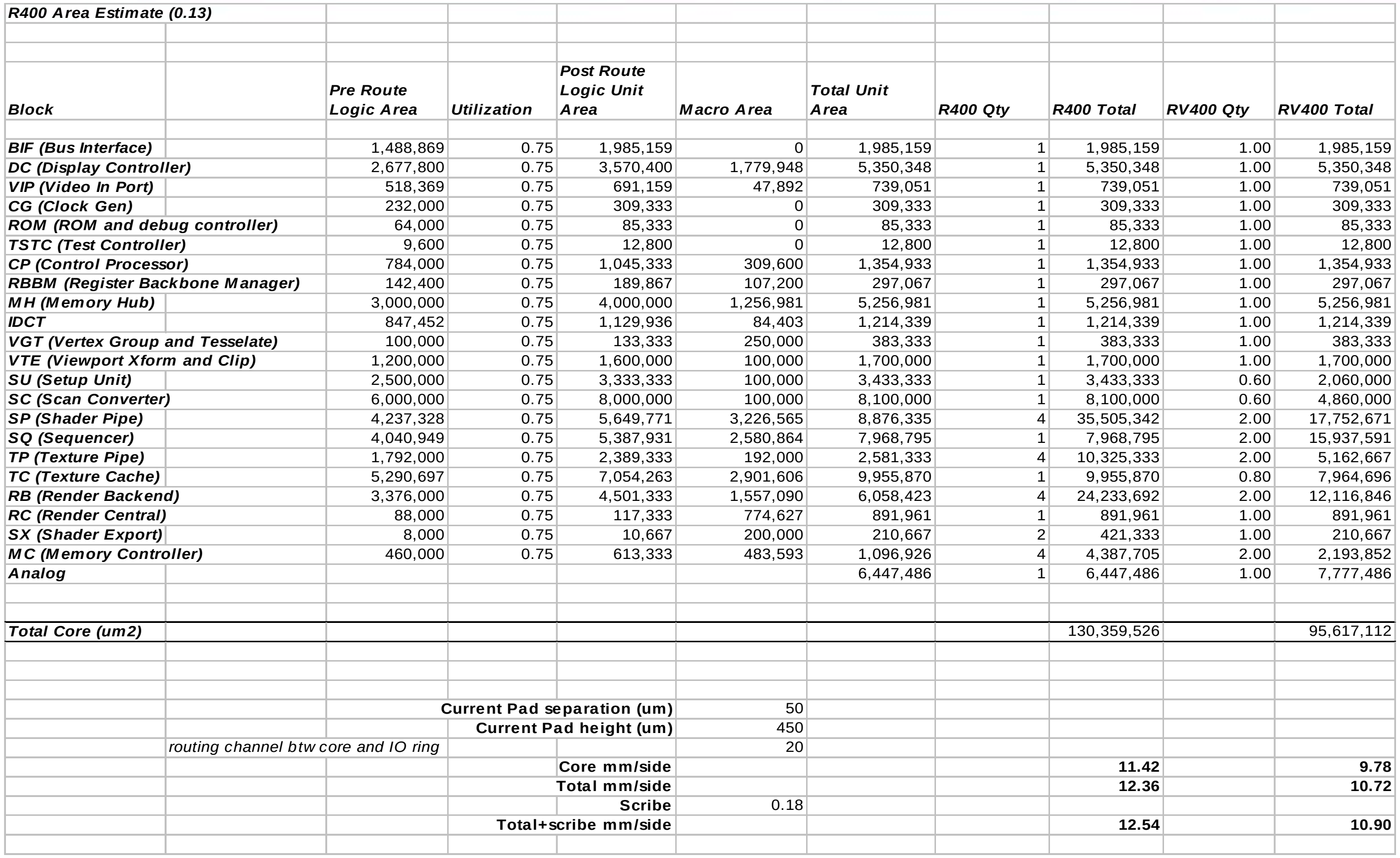
R400 Executive Review

❖ Opening Statement	Robert Feldstein
❖ Introduction	Peter Pellerite
❖ Marketing Overview	Andy Thompson
❖ Architecture	
• 3D Top Level / Performance	Steve Morein
• Display Architecture	David Glen
• SW Architecture	Phil Rogers
❖ Area Estimate / Scalability	Peter Pellerite
❖ Lunch	
❖ Program Schedule	Peter Pellerite
• Software Schedule	Kerry Wilkinson
❖ RV450	Peter Pellerite
❖ Wrap-up	Peter Pellerite

Area Estimate

Assumptions

❖ Routing efficiency	75%
❖ Pad size	50µ x 450µ
❖ R400 in 0.13µ	
➢ Core Size	11.4mm
➢ Total area	12.5mm
❖ RV450 in 0.10	
➢ Core Size	7.9mm
➢ Total Area	9.0mm



R400 Area Estimate (0.13)										
Block		Pre Route Logic Area	Utilization	Post Route Logic Unit Area	Macro Area	Total Unit Area	R400 Qty	R400 Total	RV400 Qty	RV400 Total
BIF (Bus Interface)		1,488,869	0.75	1,985,159	0	1,985,159	1	1,985,159	1.00	1,985,159
DC (Display Controller)		2,677,800	0.75	3,570,400	1,779,948	5,350,348	1	5,350,348	1.00	5,350,348
VIP (Video In Port)		518,369	0.75	691,159	47,892	739,051	1	739,051	1.00	739,051
CG (Clock Gen)		232,000	0.75	309,333	0	309,333	1	309,333	1.00	309,333
ROM (ROM and debug controller)		64,000	0.75	85,333	0	85,333	1	85,333	1.00	85,333
TSTC (Test Controller)		9,600	0.75	12,800	0	12,800	1	12,800	1.00	12,800
CP (Control Processor)		784,000	0.75	1,045,333	309,600	1,354,933	1	1,354,933	1.00	1,354,933
RBBM (Register Backbone Manager)		142,400	0.75	189,867	107,200	297,067	1	297,067	1.00	297,067
MH (Memory Hub)		3,000,000	0.75	4,000,000	1,256,981	5,256,981	1	5,256,981	1.00	5,256,981
IDCT		847,452	0.75	1,129,936	84,403	1,214,339	1	1,214,339	1.00	1,214,339
VGT (Vertex Group and Tesselate)		100,000	0.75	133,333	250,000	383,333	1	383,333	1.00	383,333
VTE (Viewport Xform and Clip)		1,200,000	0.75	1,600,000	100,000	1,700,000	1	1,700,000	1.00	1,700,000
SU (Setup Unit)		2,500,000	0.75	3,333,333	100,000	3,433,333	1	3,433,333	0.60	2,060,000
SC (Scan Converter)		6,000,000	0.75	8,000,000	100,000	8,100,000	1	8,100,000	0.60	4,860,000
SP (Shader Pipe)		4,237,328	0.75	5,649,771	3,226,565	8,876,335	4	35,505,342	2.00	17,752,671
SQ (Sequencer)		4,040,949	0.75	5,387,931	2,580,864	7,968,795	1	7,968,795	2.00	15,937,591
TP (Texture Pipe)		1,792,000	0.75	2,389,333	192,000	2,581,333	4	10,325,333	2.00	5,162,667
TC (Texture Cache)		5,290,697	0.75	7,054,263	2,901,606	9,955,870	1	9,955,870	0.80	7,964,696
RB (Render Backend)		3,376,000	0.75	4,501,333	1,557,090	6,058,423	4	24,233,692	2.00	12,116,846
RC (Render Central)		88,000	0.75	117,333	774,627	891,961	1	891,961	1.00	891,961
SX (Shader Export)		8,000	0.75	10,667	200,000	210,667	2	421,333	1.00	210,667
MC (Memory Controller)		460,000	0.75	613,333	483,593	1,096,926	4	4,387,705	2.00	2,193,852
Analog						6,447,486	1	6,447,486	1.00	7,777,486
Total Core (um2)								130,359,526		95,617,112
				Current Pad separation (um)	50					
				Current Pad height (um)	450					
	routing channel btw core and IO ring				20					
				Core mm/side				**11.42**		**9.78**
				Total mm/side				**12.36**		**10.72**
				Scribe	0.18					
				Total+scribe mm/side				**12.54**		**10.90**

Block estimates

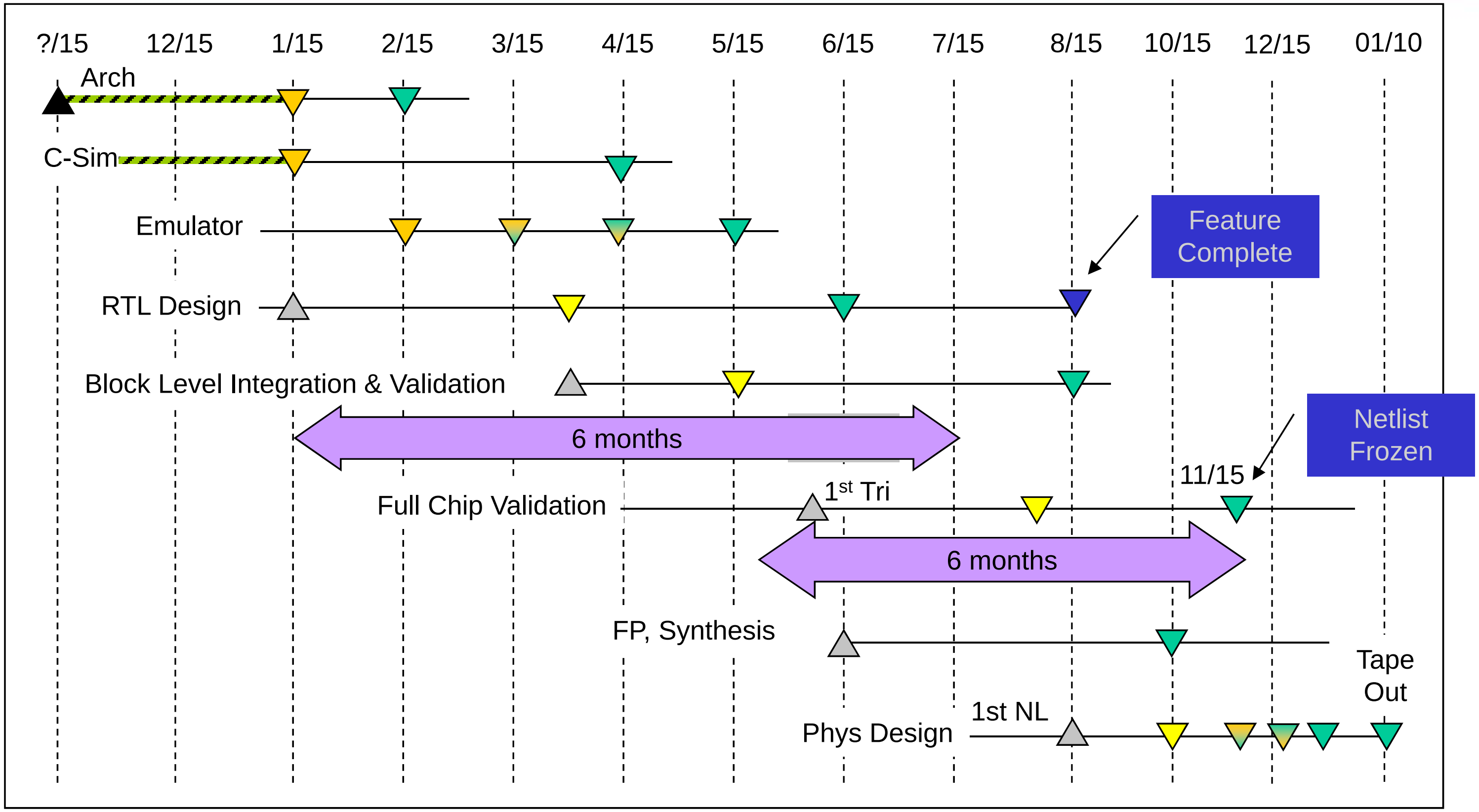
R400 Program Schedule

Task	Plan	Actual	Forecast
Significant Architecture Issue Identification Complete	10-15-01	10-10-01	
Significant Architectural Issues Resolved	12-17-01	12-20-01	
Emulator Test template Complete	01-18-02		01-18-02
PrimLib has calls for basic tests	02-01-02		02-01-02
Updated Area Estimate Complete / Specs Complete	02-15-02		02-15-02
GC Emulator integration – 1 triangle	02-22-02		02-22-02
Core Emulator pixel / shader tests	03-15-02		03-15-02
Block Testing Begins	04-16-02		04-16-02
GC/Chip Integration Start	05-17-02		05-17-02
Simulate 1 Triangle / Emulator ready for SW	06-03-02		06-03-02
First Syntheses	07-12-02		07-12-02
Verilog Feature Complete	09-16-02		09-16-02
IKOS Emulation start	10-11-02		10-11-02
Synthesis meets Timing	10-25-02		10-25-02
Verilog Frozen	11-08-02		11-08-02
IKOS Emulation (w/ Software) begins	11-11-02		11-11-02
Final Netlist (Gate level ECO only)	11-15-02		11-15-02
A11 Base Layers Tapeout	01-10-03		01-10-03
A11 Metal Layers Tapeout	01-24-03		01-24-03
First Samples for engineering evaluation	03-12-03		03-12-03
A12, A13 Netlist			
A12, A13 Samples			
Volume Ramp			
Product Delivery	July ‘03		July ‘03

CONFIDENTIAL

January 17th, 2002

R-400 Top Level Schedule



- Yellow Milestones 50-70% Complete
- Green Milestones represent >90% Complete

CONFIDENTIAL

January 17th, 2002

R200 / R400 Comparison

- Technology
  - R200 with 0.15µ technology, presents similar risk as R400 in 0.10µ technology
  - R400 in 0.13µ (Artisan Libraries and Virage macros will be proven in RV350)
  - R400 move to 0.13HS+ from 0.13LV similar to R200 switch from 0.15G to 0.15LV
  - R400 in 0.13µ reduces risk from overall R200 schedule
- Analog IP
  - R200 re-used the same DAC / PLL as R100
    - R200 re-use incorporated level shifters, not required for R400
  - R400 plan is to re-use the R300/RV350 analog designs to reduce risk
    - In parallel, R400 will explore alternate PLL design (possible area savings)
    - Test chip required
- Pads
  - R200 re-used R100 pads (incorporated level shifters)
  - R400 re-using RV350 pads
    - Plan of record is to re-layout pads - small risk relative to R200
- Overall
  - Technology / IP / Pad risk is lower than the R200 experience

R200 / R400 Comparison

❑ Teams

- R200
  - 3D was staffed with 36 resources
  - 21 logic design, 5 emulation, 10 verification
  - The R200 team was significantly distracted by R100 bring-up (Spring ’00)
  - The R200 team was a relatively junior team
    - R200 experienced significant turnover in Summer ’00 (Marlborough)
- R400
  - 3D is staffed with 51 resources
  - 26 logic design, 10 emulation, 11 verification, 4 system integration
  - Plan to add 1 logic designer, 2 verification, 1 system integration
    - Engineers needed on RL250 design; this will require backfill on the R400
  - R400 is a substantially more experienced team

CONFIDENTIAL

January 17th, 2002

R200 / R400 Comparison

❑ Test Environment

- R200
  - Slow, unstable environment for regression testing
    - Required 7 days to complete the full regression suite
  - Limited block level testing, depended on full chip simulation testing
- R400
  - Creating faster environment for more rapid spins through regression testing
    - Shift to Linux
    - Shift to VCS Verilog
  - More bugs can be found and corrected in the same amount of time
  - R400 will emphasize much greater block level testing to rapidly fix bugs in the block before full chip simulation testing
    - Moving towards R300 model of block testing

R200 / R400 Comparison

❑ Physical Design

- R200
  - R200 Physical design was time intensive (Dec. ’00 to Mar, ’01)
  - Front-end deliverables to physical design were not optimal
    - Top level signals were not constrained well
    - Too much focus on block level timing without chip level analysis
    - Chip level timing performed late - difficult to react without schedule impact
- R400
  - Physical design cycle expected to be shorter
  - Logic Design will deliver higher quality netlist
    - Better top level constraints on all signals will be used
    - Full chip timing analysis to be done pre-route
  - Allows faster convergence on issues
  - Will review RV250 and R300 experiences and leverage any additional experience for the R400

Physical Design Schedule

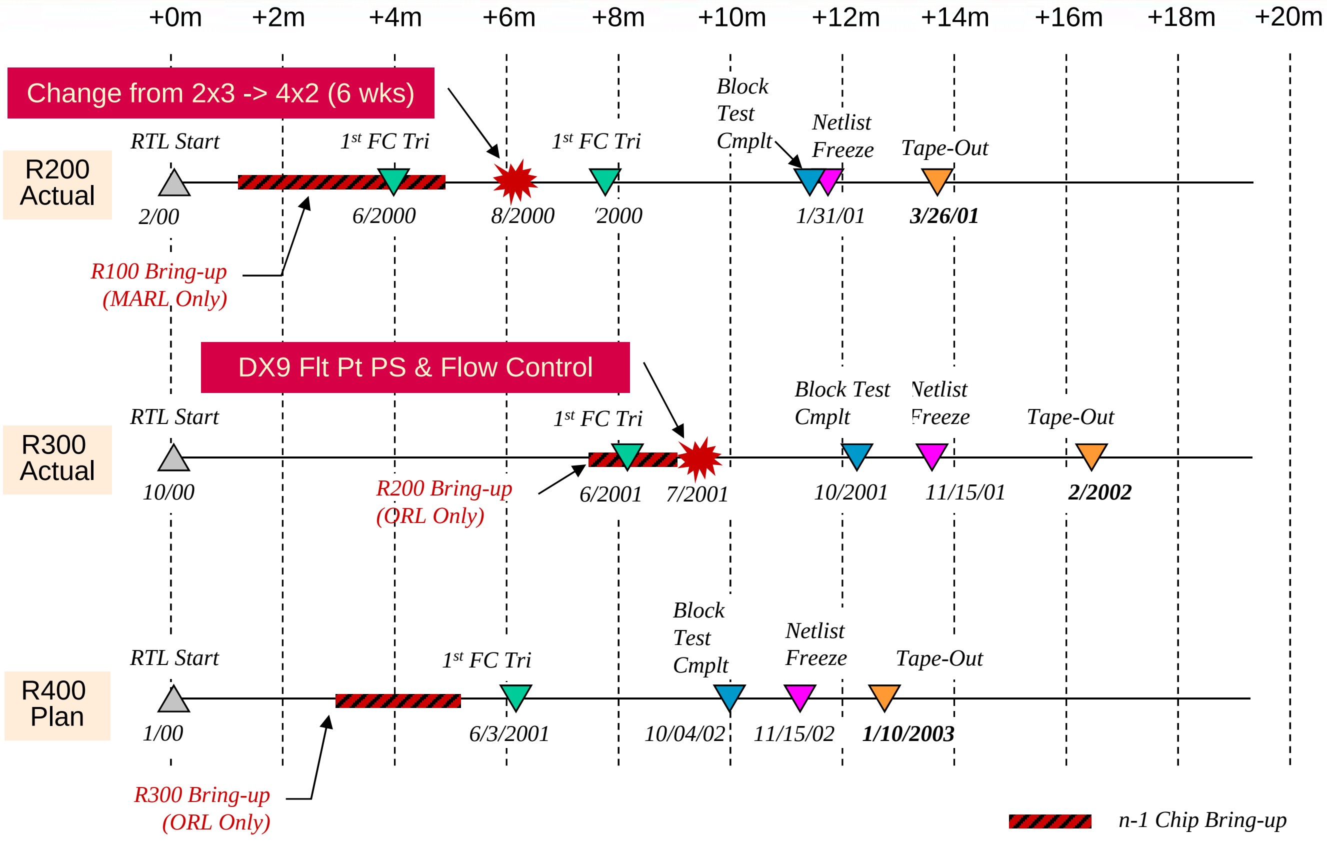
Task	Plan	Actual	Forecast
RV250/R300 PD post mortem	03/01/02		03/01/02
New Standard cells defined	03/15/02		03/15/02
R400 PD methodology proposal	04/01/02		04/01/02
Physical Architect in Marlboro	05/01/02		05/01/02
R400 PD methodology in place	06/01/02		06/01/02
New standard cells available	06/15/02		06/15/02
R400 PD methodology initial test complete	07/01/02		07/01/02
2 P&R engineers in Marlboro	07/12/02		07/12/02
Initial Synthesis complete	07/12/02		07/12/02
Initial Floorplan ready	07/12/02		07/12/02
Initial feedback from PD to logic designers	08/01/02		08/12/02
Additional P&R resources available	09/15/02		09/15/02
Final netlist release	11/15/02		11/15/02
A11 Base Tapeout	01/10/03		01/10/03

R200 / R400 Comparison

❑ Schedule Risk

- R200 / R400: Start to Simulate 1st Triangle
  - Technology / IP / Pad risk balances out (R400 is similar to lower risk as R200)
  - R400’s larger, more experienced team will enable more rapid code development and debug
  - Block level regression environment better (more rapid debug of blocks)
- R200 / R400: 1st Triangle to Tape out
  - R200 spent nine months from 1st triangle to tape out
    - An architecture switch caused a 1.5 month delay
    - The corrected duration is 7.5 months for R200 1st triangle to tape out
  - R400 plans a similar amount of time, 7.5 months, but ....
  - Better, faster simulation environment allows faster turns
  - Back-end process will proceed more smoothly
    - Chip physical design/layout and fully staffed team
    - A system integration team driving constraints for front end to back end timing

Program Comparison



+0m +2m +4m +6m +8m +10m +12m +14m +16m +18m +20m

Change from 2x3 -> 4x2 (6 wks)

R200 Actual

RTL Start — 2/00

1st FC Tri — 6/2000

8/2000

1st FC Tri — '2000

Block Test Cmplt

Netlist Freeze — 1/31/01

Tape-Out — **3/26/01**

R100 Bring-up (MARL Only)

DX9 Flt Pt PS & Flow Control

R300 Actual

RTL Start — 10/00

R200 Bring-up (ORL Only)

1st FC Tri — 6/2001

7/2001

Block Test Cmplt — 10/2001

Netlist Freeze — 11/15/01

Tape-Out — **2/2002**

R400 Plan

RTL Start — 1/00

1st FC Tri — 6/3/2001

Block Test Cmplt — 10/04/02

Netlist Freeze — 11/15/02

Tape-Out — **1/10/2003**

R300 Bring-up (ORL Only)

▨ n-1 Chip Bring-up

R200 / R400 Comparison

❑ Current Status

- IDCT, BIF, VIP completed verilog translation and passed formal verification. Regression in progress
- DC has closed most interface specs. Block specs in progress
- Emulation environment available to debug shader code
- SP/SQ, RB/RC/SX, MC/MH, PA/CP/RBBM on-track towards a 1st triangle in GC emulation
  - 1ST Triangle through emulator milestone 2/22: Forces Emulator integration
  - Vertex & Pixel Shaders 3/22

Risks / Issues / Mitigation

- Physical Design Methodology
  - Work with R300/RV250 team to understand lessons learned
  - Work with Raj Verma & Greg Buchner to develop quick iteration back-end flow
- Validation Resources (Test Generation / Test Environment)
  - Develop Plan (R-400 Mgt Team)
- Software MM team must thourougly integrate into R400 software architecture team
  - Action for R-400 Mgmt Team
- R400 API Exposure
  - Develop Plan for exposure in DX9.1 (Arch & ARG teams)
  - Develop tools for ISV’s (ARG)

RV450 Plan

- ❑Resource plan for implementing the R450/RV450
    - oRL250 team will roll onto the RV450 team
    - oR400 team will split:
        - ▪One portion will focus on debug and metal spins for the R400
        - ▪The other portion will focus on performance bottlenecks
            - •This team will tie in with the S/W performance team
    - oThe performance hardware team will implement the R450
- ❑The RV450 will be done first, the R450 second
    - oRV450 team will drive 0.1um technology
    - oR450 will leverage 0.1um technology developed for the RV450

RV450 Schedule

•R400 A11 Tapeout	01/10/03
•R400 A11 Samples	03/07/03
•R400 Performance Analysis and Debug	05/16/03
•RV450 Logic Design Complete	07/11/03
•RV450 A11 Tapeout	09/05/03
•RV450 A11 Samples	10/31/03
•RV450 Shipped	02/27/04

1st Triangle Definition
(6/3 Eng Goal; 6/15 Company Goal)

- 3D Triangle
- Indices via Command Stream
- No Clip
- No Texture
- No Z
- Fetching Vertices via Texture Path
- Vertex / Pixel Shader

- Blocks Included
  - CP / RBBM, MC, MH, VGT, PA, SC, SQ, SX, SP, RB, RC, TC, TP
- Not Included
  - BIF, DC (VGA, VIP), IDCT
  - Set State Packets not Required
  - Context Switching not Required

Logic Design Status

- Verilog started on all blocks

Block	RTL (*)	BLT (**)	
SQ	30-Apr	22-May	
SP	22-Mar	22-May	
SX	30-Apr	22-May	
RC	22-Apr	22-May	
RB	22-Apr	22-May	
TP	22-Apr	22-May	
TC	30-Apr	22-May	
MH	31-Mar	22-May	
MC	30-Apr	22-May	
* Enough RTL for simple triangle functionality			
** Simple triangle passing block level test			

Emulator Status

- Simple triangle test ran on emulator using Primlib calls on (2/22).
- Simple textured triangle ran on emulator using Primlib calls (3/19).
- Simple Z buffered triangle to run on (3/22).
- Next GC goal, on 4/15 –Multiple contexts, trilinear/anisotropic filtering, alpha blending

CONFIDENTIAL

Design Verification

- Ferret based SP testbench running on simple triangle test.
- GC.v wrapper being worked on (target compilation by 3/31)
- Target for remaining test benches (SQ/SP/SX, RC/RB, TP/TC, MH/MC all running by 5/22)

CONFIDENTIAL

Risks / After 6/15

- SQ exposure due to change in DX10 Pixel Shader Control Flow Model
  - MS not likely to be receptive to continuation of clause/phase exposure in API
  - Team to complete minimal SQ to allow for extensive GC testing before going on to new version

CONFIDENTIAL

Risks / After 6/15

- Verification resources
  - Thin
  - New hire learning curve
- Post 6/15 Milestones
  - Z buffering, HiZ, Multi-sample (RC/RB)
  - Texture functionality (TP/TC)

CONFIDENTIAL

R400 Program Review

March 22, 2002

CONFIDENTIAL

R-400 WorkShare

- Graphics Core Big Back End – MARLBORO
  - SQ, SP, SX, RC, RB, TP, TC, MH, MC, ROM
- Graphics Core Front End – ORLANDO
  - VGT, PA, SC, CP, RBBM
- DISP, BIF, IDCT - Toronto

CONFIDENTIAL

Logic Design Status

Component	Architecture	Design	1st Tri Ready	Emulation
PA (Primitive Assembly)	Green	Green	21-May	Green
SC (Scan Converter)	Green	Green	20-May	Green
VGT (Vertex Grouper Tess.)	Green	Green	24-May	Green
CP (Command Processor)	Yellow	Yellow	15-Jun	Green
RBBM (Register Backbone)	Green	Green	1-May	Green

- On Track for 1st Triangle 6/15
- Command Processor is ORL critical path
  - Adding staff to mitigate

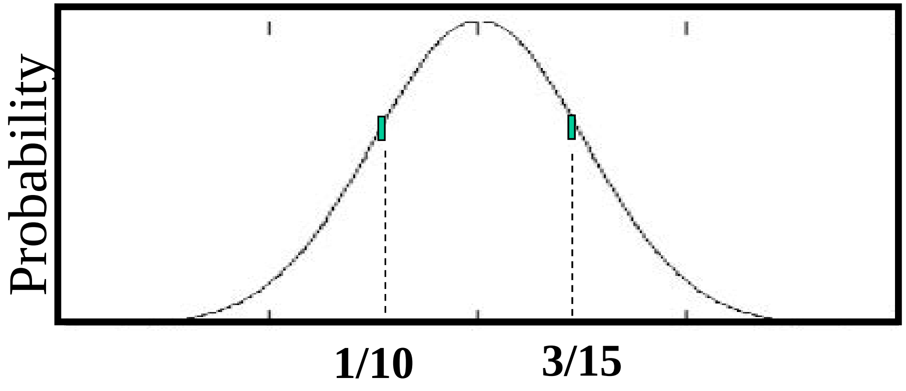
Test Plan / Generation

- Test Plan calls for 1183 tests

Vertex Grouper / Tessellator:	276 tests
Primitive Assemble:	680 tests
Scan Conversion:	356 tests
CP / RBBM	??? tests

- Estimated staff for VGT, PA, SC is 5 persons
  - Two full time engineers assigned
  - Using a Summer intern and architects to fill remaining staff
- CP test plan by mid May
  - Team (4) + 1 to implement
- Randoms GC Owner Needed to establish conceptual approach
  - Working to identify resource

R-400 Tape-Out Risk



Design Phase	Time	<50% Date
Block Design	6 months	12/15/2001
1st Triangle	milestone	6/15/2002
Chip Verification	22 weeks	11/15/2002
Physical Design	8 weeks	1/15/2003

Design Phase	Time	80% Date
Block Design	6 months	12/15/2001
1st Triangle	milestone	6/15/2002
Chip Verification	28 weeks	12/31/2002
Physical Design	10 weeks	3/15/2003

- On Track for 6/15 1st Triangle
- Jan 10th remains possible but % confidence not high
- Mar 15th is a reasonable 80% confidence target
- Actively Managing Team to Jan 10th

CONFIDENTIAL

Orlando Risk Summary

- R300 Bring-up
  - Planned bring-up staff
    - 11 Software (5 OGL, 6 DX)
    - 1 Diag / HW Engineer (Silver)
    - .5 Physical Design (Camlin)
  - If additional help required, then R400 schedule risk.
- Validation Test Suite Generation currently under-staffed
- Post 6/15 Milestones
  - Synchronization scheme
  - Real-Time Streams
  - Remaining 2D Packets (HostData BLT & BitBLT required for 1st Triangle)
  - System Performance
  - Area Optimization

CONFIDENTIAL

Backup Slides

CONFIDENTIAL

Post 1st Triangle Design & Validation Priorities

- Synchronization scheme
- Real-Time Streams
- Line Stipple
- Z Accuracy Uncertainity
- Vis-Query
- Multi-Pass Operations
  - Vertex Shaders
  - Pixel Shaders
  - Indirect Buffers

- Flexible Flow Control w/ dependent fetching
- Remaining 2D Packets (HostData BLT & BitBLT required for 1st Triangle)
- System Performance
- Area Optimization
- Timing
- Bad Pipe Disable
- Distributed Dynamic Clocking
- Shadow Mapping (Mark)

CONFIDENTIAL

ATI TECHNOLOGIES INC.

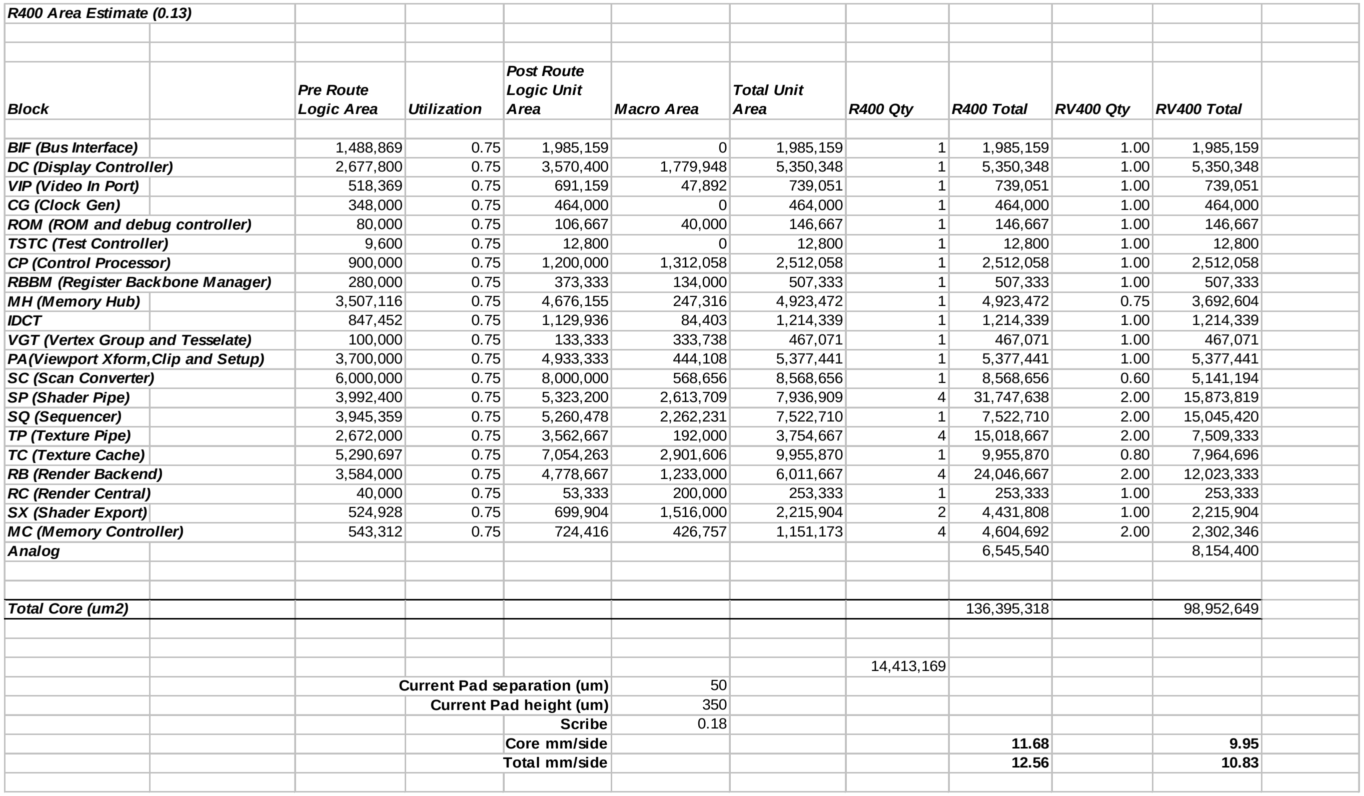
R400 Program – March Review

ATI 2060
LG v. ATI
IPR2015-00326

Area Estimate

Assumptions

• Routing efficiency	75%
• Pad size	50µ x 350µ
• R400 in 0.13µ	
• Core Size	11.7mm
• Total area	12.6mm
• RV450 in 0.10µ (estimated)	
• Core Size	8.0mm
• Total Area	8.9mm



R400 Area Estimate (0.13)

Block	Pre Route Logic Area	Utilization	Post Route Logic Unit Area	Macro Area	Total Unit Area	R400 Qty	R400 Total	RV400 Qty	RV400 Total
BIF (Bus Interface)	1,488,869	0.75	1,985,159	0	1,985,159	1	1,985,159	1.00	1,985,159
DC (Display Controller)	2,677,800	0.75	3,570,400	1,779,948	5,350,348	1	5,350,348	1.00	5,350,348
VIP (Video In Port)	518,369	0.75	691,159	47,892	739,051	1	739,051	1.00	739,051
CG (Clock Gen)	348,000	0.75	464,000	0	464,000	1	464,000	1.00	464,000
ROM (ROM and debug controller)	80,000	0.75	106,667	40,000	146,667	1	146,667	1.00	146,667
TSTC (Test Controller)	9,600	0.75	12,800	0	12,800	1	12,800	1.00	12,800
CP (Control Processor)	900,000	0.75	1,200,000	1,312,058	2,512,058	1	2,512,058	1.00	2,512,058
RBBM (Register Backbone Manager)	280,000	0.75	373,333	134,000	507,333	1	507,333	1.00	507,333
MH (Memory Hub)	3,507,116	0.75	4,676,155	247,316	4,923,472	1	4,923,472	0.75	3,692,604
IDCT	847,452	0.75	1,129,936	84,403	1,214,339	1	1,214,339	1.00	1,214,339
VGT (Vertex Group and Tesselate)	100,000	0.75	133,333	333,738	467,071	1	467,071	1.00	467,071
PA(Viewport Xform,Clip and Setup)	3,700,000	0.75	4,933,333	444,108	5,377,441	1	5,377,441	1.00	5,377,441
SC (Scan Converter)	6,000,000	0.75	8,000,000	568,656	8,568,656	1	8,568,656	0.60	5,141,194
SP (Shader Pipe)	3,992,400	0.75	5,323,200	2,613,709	7,936,909	4	31,747,638	2.00	15,873,819
SQ (Sequencer)	3,945,359	0.75	5,260,478	2,262,231	7,522,710	1	7,522,710	2.00	15,045,420
TP (Texture Pipe)	2,672,000	0.75	3,562,667	192,000	3,754,667	4	15,018,667	2.00	7,509,333
TC (Texture Cache)	5,290,697	0.75	7,054,263	2,901,606	9,955,870	1	9,955,870	0.80	7,964,696
RB (Render Backend)	3,584,000	0.75	4,778,667	1,233,000	6,011,667	4	24,046,667	2.00	12,023,333
RC (Render Central)	40,000	0.75	53,333	200,000	253,333	1	253,333	1.00	253,333
SX (Shader Export)	524,928	0.75	699,904	1,516,000	2,215,904	2	4,431,808	1.00	2,215,904
MC (Memory Controller)	543,312	0.75	724,416	426,757	1,151,173	4	4,604,692	2.00	2,302,346
Analog							6,545,540		8,154,400
Total Core (um2)							136,395,318		98,952,649
						14,413,169			
			Current Pad separation (um)	50					
			Current Pad height (um)	350					
			Scribe	0.18					
			Core mm/side				**11.68**		**9.95**
			Total mm/side				**12.56**		**10.83**

Block estimates

R400 Program Schedule

Task	Plan	Actual	Forecast
Significant Architecture Issue Identification Complete	10-15-01	10-10-01	
Significant Architectural Issues Resolved	12-17-01	12-20-01	
Emulator Test template Complete	01-18-02	01-18-02	
PrimLib has calls for basic tests	02-01-02	02-08-02	
Updated Area Estimate Complete / Specs Complete	02-15-02	02-18-02	
GC Emulator integration – 1 triangle	02-22-02	02-21-02	
Core Emulator pixel / shader tests run	03-15-02	03-19-02	
Block Testing Begins	04-16-02		04-16-02
GC/Chip Integration Start	05-17-02		05-17-02
Simulate 1 Triangle / Emulator ready for SW	06-03-02		06-03-02
First Syntheses	07-12-02		07-12-02
Verilog Feature Complete	09-16-02		09-16-02
IKOS Emulation start	10-11-02		10-11-02
Synthesis meets Timing	10-25-02		10-25-02
Verilog Frozen	11-08-02		11-08-02
IKOS Emulation (w/ Software) begins	11-11-02		11-11-02
Final Netlist (Gate level ECO only)	11-15-02		11-15-02
A11 Base Layers Tapeout	01-10-03		01-10-03
A11 Metal Layers Tapeout	01-24-03		01-24-03
First Samples for engineering evaluation	03-12-03		03-12-03
A12, A13 Netlist			
A12, A13 Samples			
Volume Ramp			
Product Delivery	July '03		July '03

R400 Risks / Mitigation

- DX10 Shader Control Flow Model
  - Microsoft rejecting proposed model –
  - Asking for more general purpose – longer-living model
  - Team will complete minimum SQ to support early validation, before starting new SQ
  - Working to minimize impact to overall schedule (only SQ team affected)
- Design Verification Uncertainty / Resources
  - Resources being examined
  - Proactively looking 3 months ahead.
- Physical Design
  - Issues under examination
- Software resources, Multi-Media
- Overall impact not additive
  - Continue driving the program to January 10th.

ATI TECHNOLOGIES INC.

R400 Program – GC Design Status

May 30th, 2002

Steps to 1st Triangle

DONE 1. Primlib does back door memory fill of command buffer and vertex buffer

DONE 2. (RBBM -> CP) Primlib writes regs to start CP

In Proc 3. (CP -> MH/MC) CP issues read req for command buffer

In Proc 4. (MH/MC -> CP) MH/MC returns data to CP

In Proc 5. (CP -> RBBM) CP sends regs, instructions, constants to RBBM followed by draw initiator

DONE 6. (RBBM -> blocks, RBBM -> VGT) RBBM forwards to above values to blocks, initiator to VGT

DONE 7. (VGT -> SQ) VGT sends vector of indices to GPR's

Steps to 1st Triangle (continued)

DONE 8. (SQ -> SP) SQ sends vector of indices to SP

DONE 9. (SQ -> SP) SQ tells SP to write indices to GPR's

In Proc 10. (SQ/SP -> TP/TC) SQ executes vertex shader sends fetch req to TP/TC, SP sends indices to TP

11. (TP/TC -> MH/MC) TP/TC sends read request to for vertex data to MC/MH

12. (MC/MH -> TP/TC) MC/MH returns vertex data to TP/TC

13. (TP/TC -> SP/SQ) TP/TC returns vertex data to SQ/SP

Steps to 1st Triangle (continued)

14. (SQ/SP -> SX) Vertex shader exports position and color
15. (SX -> PA) Position to PA
16. (PA -> SC) PA does setup and sends result to SC
17. (SC -> RB) SC sends tile info to RB
18. (RB -> SC) returns tile results to SC
19. (SC -> SQ/SP/SX) SC sends quad data to SQ/SP, quad address to SX
20. (SX -> SP) SP interpolates color data from SX

Steps to 1st Triangle (continued)

21. (SQ -> SP) SQ tells SP to put quad data into GPR’s
22. (SQ/SP -> SX) SQ starts pixel shader and exports color to SX
23. (SX -> RB) SX sends color data to RB
24. (RB -> MC) RB writes color data to MC

Status of first triangle

- Estimate 6/30 based on where we are
  - Changing SQ to new CF
    - Didn’t allow for prior testing of integrated SQ/SP/SX
    - Overall benefit to program (switch now rather then later)
    - SW team wanted new SQ in emu, so we didn’t want to mismatch emu and hw
  - SC team impacted by R300 – See Orlando Status

RTL Block Status

Block	Percent Coded
SP	85%
SQ	80%
SX	80%
TC	55%
TP	40%
RB	25%
MH	70%
MC	60%

RTL Milestones

7/15 - Full chip netlist to PD with:

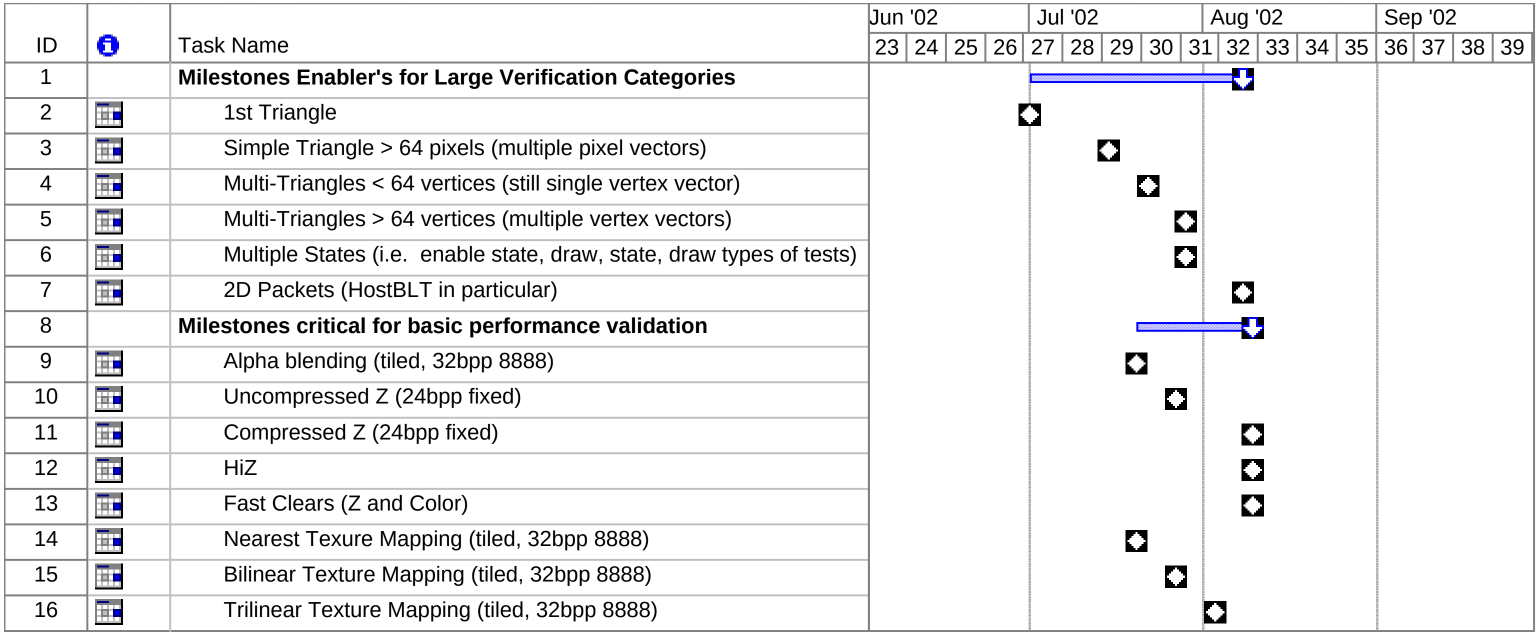
- Accurate top level interconnect
- All macros
- At least one block (SP) in “good shape”

CONFIDENTIAL

Orlando – First Triangle Status

Block	1st Tri Ready	Block % Coded	Comments
CP/RBBM	6/3	65%	• MicroCode written • Debugging problem with loading uC
VGT	5/17	88%	• Supporting 1st Tri Integration. • VGT tasks remaining: Index DMA & Reuse
PA	6/3	89%	• All Sub-blocks Complete, wit exception of some clipping functions (AG) • Integrating PA this week • Some impact due to R300
SC	6/10	65%	• Entire SC Team consumed for 2 weeks for R300 Hangs (not yet resolved) • Sub-Blocks not complete: o Quad Packer o Z Calc (including Precision changes)

Next Major Integration Milestones Goals



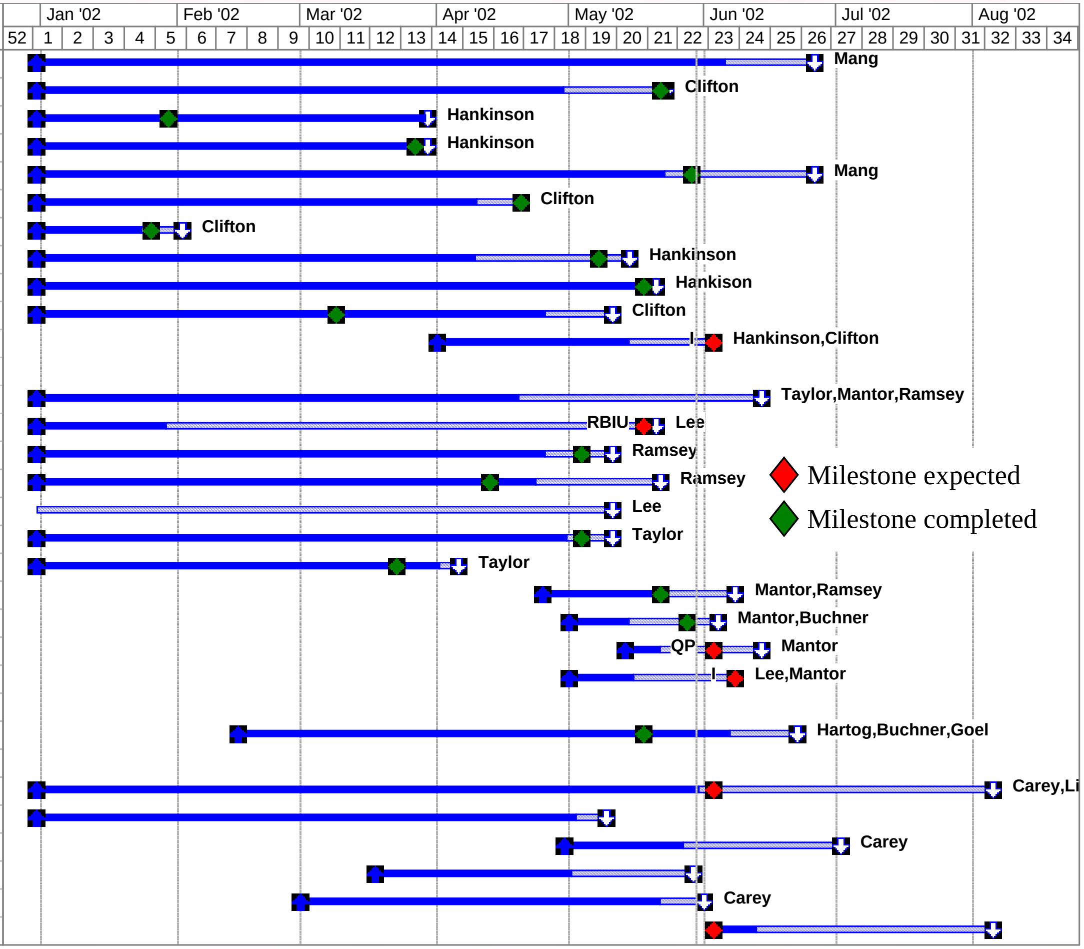
ID	ℹ	Task Name
1		**Milestones Enabler's for Large Verification Categories**
2	▦	1st Triangle
3	▦	Simple Triangle > 64 pixels (multiple pixel vectors)
4	▦	Multi-Triangles < 64 vertices (still single vertex vector)
5	▦	Multi-Triangles > 64 vertices (multiple vertex vectors)
6	▦	Multiple States (i.e. enable state, draw, state, draw types of tests)
7	▦	2D Packets (HostBLT in particular)
8		**Milestones critical for basic performance validation**
9	▦	Alpha blending (tiled, 32bpp 8888)
10	▦	Uncompressed Z (24bpp fixed)
11	▦	Compressed Z (24bpp fixed)
12	▦	HiZ
13	▦	Fast Clears (Z and Color)
14	▦	Nearest Texure Mapping (tiled, 32bpp 8888)
15	▦	Bilinear Texture Mapping (tiled, 32bpp 8888)
16	▦	Trilinear Texture Mapping (tiled, 32bpp 8888)

- Schedule represents goals for the 6 weeks following 1st Triangle
- Meeting these goals clears way for full Validation Suite Testing

Backup

Orlando Design Schedule

ID	ℹ	Task Name	% Com
1		Primative Assembly	89%
2		RBIU State Block	83%
8		Clip Code Generator	99%
14		Clipper	94%
19		Address Generator	81%
25		Viewport Xform Engine	90%
33		CC Vector Engine	71%
40		Vertex Store	74%
46		Shader Xport Controller	98%
52		Setup Engine	88%
68		Integration	70%
73			
74		Scan Converter	67%
75		RBIU State Block	21%
81		Walker / Pre-Walk	88%
87		Quad Mask	80%
97		Zcalc	0%
103		Sample Mask	91%
109		Interpolators	94%
115		Quad Select Controller	54%
120		Detail Accumulation Controlle	41%
126		Quad Packer	24%
132		Integration	37%
138			
139		Vertex Grouper Tesselator	88%
153			
154		Command Processor / RBBM	70%
155		Architecture	95%
160		Validation Plan	42%
164		CP Design	63%
172		CP/RBBM Integration	88%
177		State Packets	17%



CONFIDENTIAL

ATI TECHNOLOGIES INC.

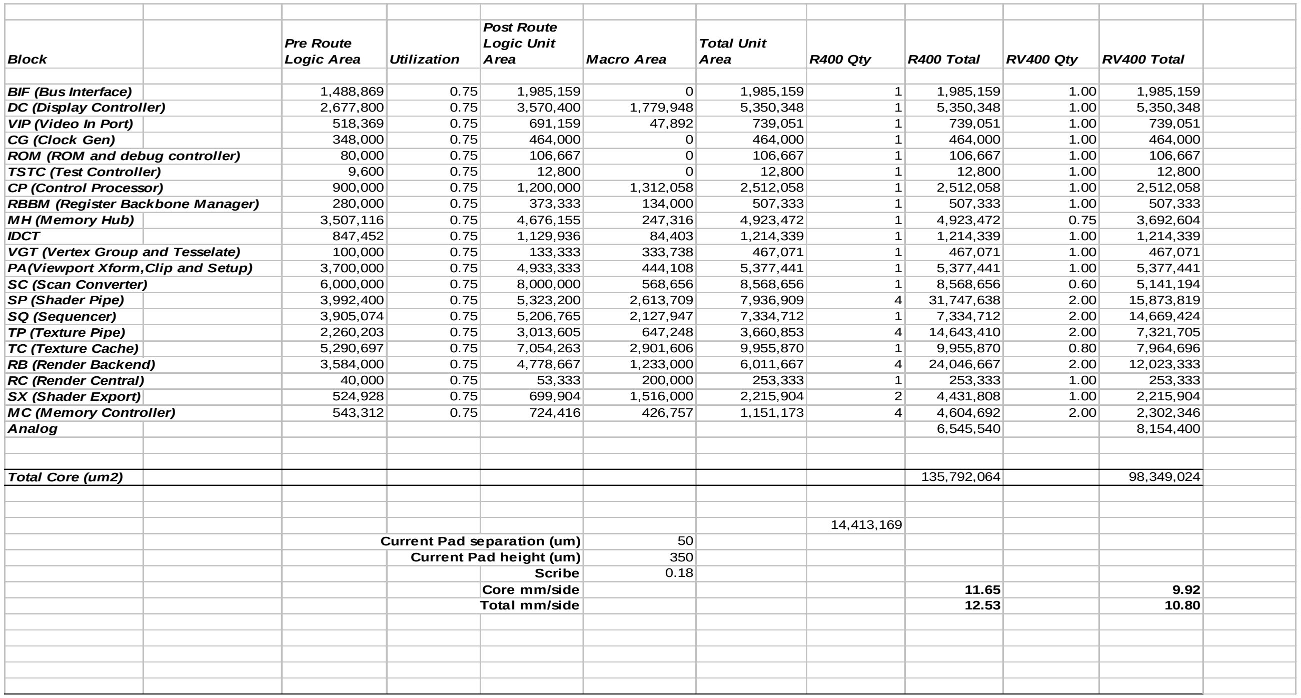
R400 Program – May Review

Area Estimate

Assumptions

- Routing efficiency 75%
- Pad size 50µ x 350µ
- R400 in 0.13µ
  - Core Size 11.7mm
  - Total area 12.6mm

- RV450 in 0.10µ (estimated)
  - Core Size 8.0mm
  - Total Area 8.9mm



Block		Pre Route Logic Area	Utilization	Post Route Logic Unit Area	Macro Area	Total Unit Area	R400 Qty	R400 Total	RV400 Qty	RV400 Total
BIF (Bus Interface)		1,488,869	0.75	1,985,159	0	1,985,159	1	1,985,159	1.00	1,985,159
DC (Display Controller)		2,677,800	0.75	3,570,400	1,779,948	5,350,348	1	5,350,348	1.00	5,350,348
VIP (Video In Port)		518,369	0.75	691,159	47,892	739,051	1	739,051	1.00	739,051
CG (Clock Gen)		348,000	0.75	464,000	0	464,000	1	464,000	1.00	464,000
ROM (ROM and debug controller)		80,000	0.75	106,667	0	106,667	1	106,667	1.00	106,667
TSTC (Test Controller)		9,600	0.75	12,800	0	12,800	1	12,800	1.00	12,800
CP (Control Processor)		900,000	0.75	1,200,000	1,312,058	2,512,058	1	2,512,058	1.00	2,512,058
RBBM (Register Backbone Manager)		280,000	0.75	373,333	134,000	507,333	1	507,333	1.00	507,333
MH (Memory Hub)		3,507,116	0.75	4,676,155	247,316	4,923,472	1	4,923,472	0.75	3,692,604
IDCT		847,452	0.75	1,129,936	84,403	1,214,339	1	1,214,339	1.00	1,214,339
VGT (Vertex Group and Tesselate)		100,000	0.75	133,333	333,738	467,071	1	467,071	1.00	467,071
PA(Viewport Xform,Clip and Setup)		3,700,000	0.75	4,933,333	444,108	5,377,441	1	5,377,441	1.00	5,377,441
SC (Scan Converter)		6,000,000	0.75	8,000,000	568,656	8,568,656	1	8,568,656	0.60	5,141,194
SP (Shader Pipe)		3,992,400	0.75	5,323,200	2,613,709	7,936,909	4	31,747,638	2.00	15,873,819
SQ (Sequencer)		3,905,074	0.75	5,206,765	2,127,947	7,334,712	1	7,334,712	2.00	14,669,424
TP (Texture Pipe)		2,260,203	0.75	3,013,605	647,248	3,660,853	4	14,643,410	2.00	7,321,705
TC (Texture Cache)		5,290,697	0.75	7,054,263	2,901,606	9,955,870	1	9,955,870	0.80	7,964,696
RB (Render Backend)		3,584,000	0.75	4,778,667	1,233,000	6,011,667	4	24,046,667	2.00	12,023,333
RC (Render Central)		40,000	0.75	53,333	200,000	253,333	1	253,333	1.00	253,333
SX (Shader Export)		524,928	0.75	699,904	1,516,000	2,215,904	2	4,431,808	1.00	2,215,904
MC (Memory Controller)		543,312	0.75	724,416	426,757	1,151,173	4	4,604,692	2.00	2,302,346
Analog								6,545,540		8,154,400
Total Core (um2)								135,792,064		98,349,024
							14,413,169			
				Current Pad separation (um)	50					
				Current Pad height (um)	350					
				Scribe	0.18					
				Core mm/side				**11.65**		**9.92**
				Total mm/side				**12.53**		**10.80**

Block estimates

R400 Program Schedule

Task	Plan	Actual	Forecast
Significant Architecture Issue Identification Complete	10-15-01	10-10-01	
Significant Architectural Issues Resolved	12-17-01	12-20-01	
Emulator Test template Complete	01-18-02	01-18-02	
PrimLib has calls for basic tests	02-01-02	02-08-02	
Updated Area Estimate Complete / Specs Complete	02-15-02	02-18-02	
GC Emulator integration – 1 triangle	02-22-02	02-21-02	
Core Emulator pixel / shader tests run	03-15-02	03-19-02	
Block Testing Begins	04-16-02	05-01-02	
GC/Chip Integration Start	05-17-02	05-15-02	
Simulate 1 Triangle / Emulator ready for SW	06-15-02		06-30-02
First Syntheses	07-12-02		07-12-02
Verilog Feature Complete	09-16-02		09-16-02
IKOS Emulation start	10-11-02		10-11-02
Synthesis meets Timing	10-25-02		10-25-02
Verilog Frozen	11-08-02		11-08-02
IKOS Emulation (w/ Software) begins	11-11-02		11-11-02
Final Netlist (Gate level ECO only)	11-15-02		11-15-02
A11 Base Layers Tapeout	01-10-03		01-10-03
A11 Metal Layers Tapeout	01-24-03		01-24-03
First Samples for engineering evaluation	03-12-03		03-12-03
A12, A13 Netlist			
A12, A13 Samples			
Volume Ramp			
Product Delivery	July ‘03		July ‘03

CONFIDENTIAL

May 30, 2002

R400 Summary

- DX10 Shader Control Flow Model Impact
  - Worked to minimize impact to overall schedule
  - Benefit to overall program to switch to new SQ control flow
- Design Verification
  - Continue to develop Verification strategy
- Physical Design
  - Flow proposal made, currently under review
  - Plan required for machines, licenses, etc.
- DFT / Chip Integration
  - R400 pads
  - Memory repair method
- Overall
  - Continuing to drive towards January 10th.

R400 Program Review

Orlando - August 30, 2002

ATI 2063
LG v. ATI
IPR2015-00326

- Design Status
- Emulator Status
- Test Suite Development Status
- Validation Status
- Synthesis Status
- Next Milestones

- Full Chip Validation

Orlando Hardware Design

- Design Complete Goal 8/31 - Commitment 9/15
  - PA Design Complete
  - VGT Design Complete
  - RBBM Design 95% Complete (Debug logic is remaining)
  - CP Design & MicroCode expected completion 9/15
    - RTL: 90% Complete -- 2D Brush Support Logic Open
    - Microcode: 95% Complete: Open is one legacy packet (Polyline) and three new 2D packets
  - SC expected completion 9/15. Open Design Items:
    - Pipe Disable (Mantor)
    - VisQuery (in Emulator; HW not large in scope)
    - XYCentroid (Mantor: in Emulator)
    - MultiPass (Clay: Emulator; Mantor HW)

Joe Cox
Slides 1 to 9

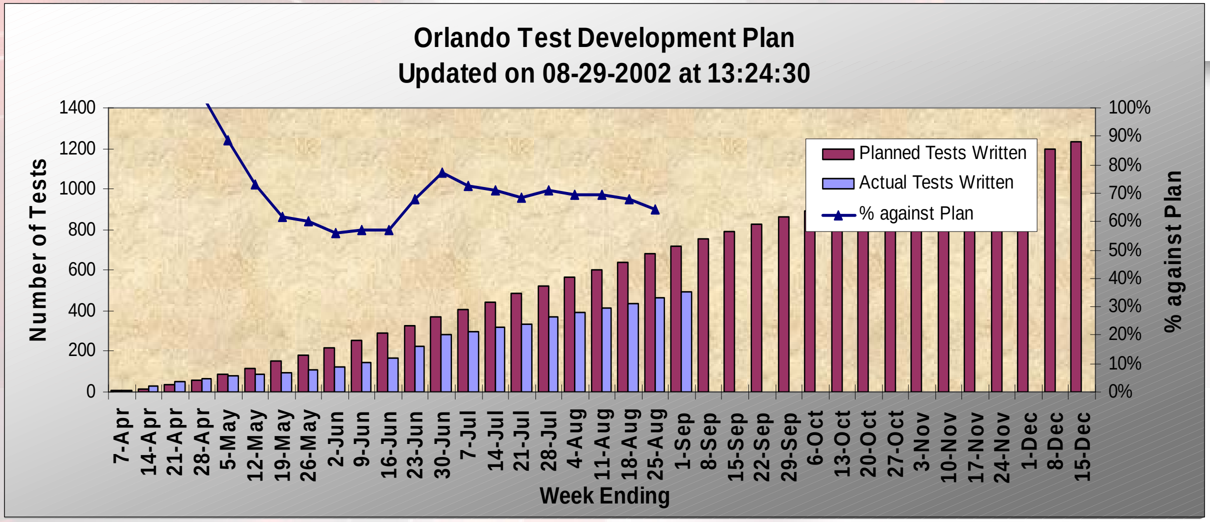
Christeen
Gray
Slides 10 to 13

CONFIDENTIAL
9/4/2015

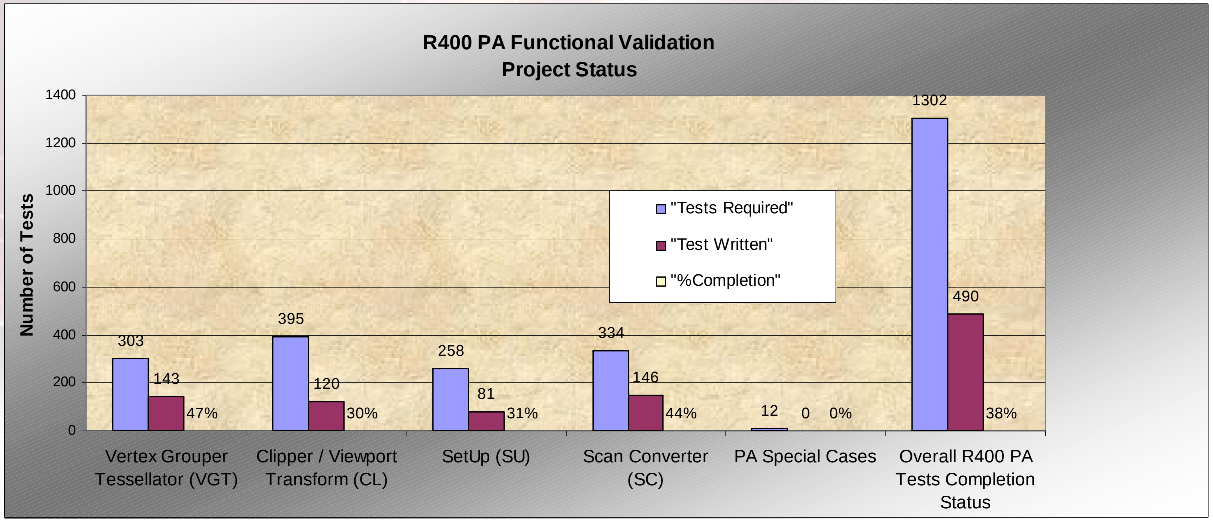
Emulator

- PA Emulator Feature Complete
- VGT Emulator Feature Complete
- CP Emulator Status
  - 2D Support is the main focus of the CP emulator at present
  - Real-Time Event Engine still to be developed (About 2 Weeks of effort)
  - DMA Engine needs to be updated (A few days of effort)
- SC Emulator Complete Except for:
  - Pipe Disable
  - MultiPass Pixel Shader

Test Suite Development



Orlando Test Development Plan
Updated on 08-29-2002 at 13:24:30



R400 PA Functional Validation
Project Status

Joe Cox
Slides 1 to 9

Christeen Gray
Slides 10 to 13

Slide 5

CONFIDENTIAL
9/4/2015

Validation

- Block Level Test Regressions

Test Bench	Tests Run	Tests Pass	% Pass	Report Date
VGT	504	503	99%	08-29-02
CP-RBBM	719	702	97%	08-29-02
SC	476	155	32%	08-28-02
PA	504	438	86%	08-29-02

- Block Level Random Testing
  - PA Block Randoms coded & running against RTL (failing ~ 10%)
  - VGT Block Randoms Code starting week of 9/2
  - SC Block Randoms Code starting week of 9/2

Joe Cox
Slides 1 to 9

Christeen
Gray
Slides 10 to 13

Slide 6

CONFIDENTIAL
9/4/2015

HOS Test Development Status

	R400 HOS Validation									
Primitive Type	**Tessellation Type**	**Reuse Depth**	**Tessellation Level**	**Position Shader Type**	**Normal Shader Type**	**Parameter Shader Type**	**Multi-pass**	**# Tests**	**Plan Date**	**Comments**
PNT	Discrete	4 to 16	1 to14 (disc), 1.0 to 15.0 (cont)	Linear, Cubic	Linear, Quadratic	Linear	No	9	7/26	**done**
	Continuous			Linear, Cubic	Linear, Quadratic	Linear	no	9	7/26	**done**
	Adaptive			Linear, Cubic	Linear, Quadratic	Linear	yes	4	9/30	test written but vertex export problem exists
PNQ	Continuous	4 to 16	1.0 to 15.0 (cont)	Linear, Cubic	Linear, Quadratic	Linear	no	5	7/26	**done**
	Adaptive			Linear, Cubic	Linear, Quadratic	Linear	yes	4	9/30	vertex export support needed
PNL	Continuous	4 to 16	1.0 to 15.0 (cont)	Linear, Cubic	Linear, Quadratic	Linear	no	3	7/26	**done**
	Adaptive			Linear, Cubic	Linear, Quadratic	Linear	yes	3	9/30	vertex export needed
R-Patch	Continuous	4 to 16	1.0 to 15.0 (cont)	Cubic	Cubic	Cubic	no	3	8/31	shader problem being fixed, tests written
	Adaptive			Cubic	Cubic	Cubic	yes	3	9/30	vertex export needed
T-Patch	Continuous	4 to 16	1.0 to 15.0 (cont)	Cubic	Cubic	Cubic	no	3	8/23	**done**
	Adaptive			Cubic	Cubic	Cubic	yes	3	9/30	vertex export needed
L-Patch	Continuous	4 to 16	1.0 to 15.0 (cont)	Cubic	Cubic	Cubic	no	2	8/9	**done**
	Adaptive			Cubic	Cubic	Cubic	yes	2.00	9/30	vertex export needed

Joe Cox
Slides 1 to 9

Christeen Gray
Slides 10 to 13

Synthesis & Timing

- Latest Synthesis Summary

Block	PA	SC	VGT	CP
Approx. Violations	~4000	~4000	~600	~2000
Worst Violations	1.05 nS	0.62 nSec	0.57 nSec	0.37 nSec
Synth Logic Area	2.655 sqmm	5.68 sqmm	0.807 sqmm	2.686 sqmm
Macro Area	0.58 sqmm	0.57 sqmm	0.332 sqmm	1.497 sqmm
Est. Post Route	4.37 sqmm	8.69 sqmm	1.485 sqmm	5.334 sqmm

Joe Cox
Slides 1 to 9

Christeen
Gray
Slides 10 to 13

CONFIDENTIAL
9/4/2015

Orlando Program Schedule

- Design Complete
  - Goal 8/31
  - Commitment 9/16
- 2nd NetList
  - Goal 9/23
- Code Freeze
  - Goal 11/18

Joe Cox
Slides 1 to 9

Christeen Gray
Slides 10 to 13

Full Chip Validation

Christeen Gray

Chip Validation Areas

TORONTO	ORLANDO	MARLBORO
BIF CRTC DCP DISPOUT IDCT LB SCL TVOUT VGA VIP	CP RBBM VGT PA SC	SQ SP SX MH/MC RB RC TC/TP CG IO TST/DBG ROM
System Tests	Real Time Streams Motion Comp. Performance System Tests	Performance System Tests

Joe Cox
Slides 1 to 9

Christeen
Gray
Slides 10 to 13

Slide 11

CONFIDENTIAL
9/4/2015

Chip Validation Status

- Toronto
  - Passing simple functional tests … attempting functional chip test in each category
  - Stubs created for GFX blocks … now can select between fullchip / partial chip configuration
- Orlando
  - Passing simple register read / write test.
  - Debugging first chip level graphic test … test passes all initialization
- Marlboro:
  - Passing simple register read / write test

Joe Cox
Slides 1 to 9

Christeen
Gray
Slides 10 to 13

Slide 12

CONFIDENTIAL
9/4/2015

Chip Validation Status (cont'd)

- Weekly meetings started on 8/16
- Locations of Block Regression Status (soon to contain Chip Regression data)

- MARLBORO:http://www.ma.atitech.com/r400/regression_reports
- ORLANDO:http://www.fl.atitech.com/R-400/BlockRegressORL.shtm
- TORONTO:http://uhw.atitech.ca/regression/crayola

Joe Cox
Slides 1 to 9

Christeen Gray
Slides 10 to 13

CONFIDENTIAL
9/4/2015

BACKUP CHARTS

CP Area Growth

- Area of CP is larger than expected.
- Plan to review the synthesis reports and make adjustments to reduce the area after we are further along with the validation.
- Main Reasons for Area Growth are:
  - Real-Time event engine added -- 16 slices
  - Pre-Fetch Parser Addition
  - Hiding 2X AGP latency as past chips (Before and After Pre-Fetch Parser)
  - Many more registers are in the CP to control its function.
  - Micro Engine Instuction RAM is larger than past chips -- Mostly because of 2D translation in microcode.

Note the microengine is not a significant

Joe Cox
Slides 1 to 9

Christeen
Gray
Slides 10 to 13

Slide 15

CONFIDENTIAL
9/4/2015

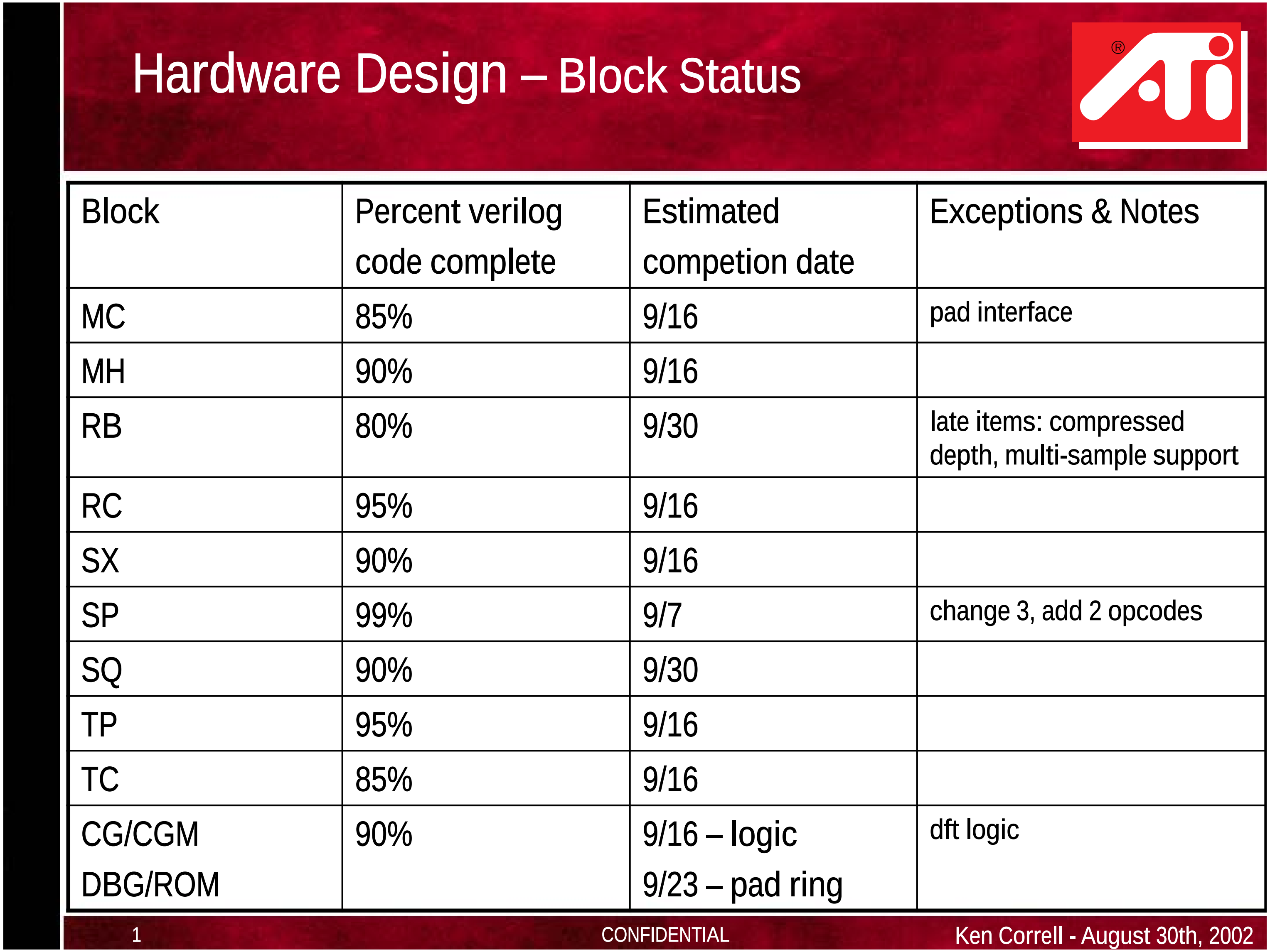
CP Validation – More Info

- Unit Level Validation : About 620 unit tests are developed and passing.
  - This includes DMA, Real-Time, 2D, Ring, and Indirect Buffers, Context Switching etc.
  - Estimate is that Unit-Testing will complete by the End of September.
- Full Chip Validation : Currently debugging tests with just the emulator
- 2D test conversion/debug is in-progress
  - one of each type of 2D test completed and have corrected many bugs.
- 20 DMA tests have been developed, some are working.
- 4 Real-Time Tests have been written and are waiting Emulator Mods to support.
- 3D test development has also begun.

Joe Cox
Slides 1 to 9

Christeen
Gray
Slides 10 to 13

CONFIDENTIAL
9/4/2015



Hardware Design – Block Status

Block	Percent verilog code complete	Estimated competion date	Exceptions & Notes
MC	85%	9/16	pad interface
MH	90%	9/16	
RB	80%	9/30	late items: compressed depth, multi-sample support
RC	95%	9/16	
SX	90%	9/16	
SP	99%	9/7	change 3, add 2 opcodes
SQ	90%	9/30	
TP	95%	9/16	
TC	85%	9/16	
CG/CGM DBG/ROM	90%	9/16 – logic 9/23 – pad ring	dft logic

1 CONFIDENTIAL Ken Correll - August 30th, 2002

Hardware Design – RB and SQ plan

- RB – determined in June that the RB was in danger of being late. Reallocated resources to bring in schedule. Currently in good shape.
- SQ – adding resource to help debug tests and to review implementation for robustness.

Hardware Design – Impacts to schedule

- Parallel development of environment, emulator and rtl design
- Late architectural changes
  - SQ changes in June (rippled across many areas)
  - TC design changes to control area

Hardware Design – Open Issues / Concerns

- Star Compiler availability
  - Determine which memory compiler to use
- 600MHz Memory Interface & Pad design
  - Continue work with Joe Macri
- Performance counters needed
  - Steve is to specify counters needed by 9/6

Hardware Design – Open Issues / Concerns (cont)

- Added hardware needed for testability
  - Get definition from Frank Hering
- Tuning needed for performance
  - Work towards getting the support in hardware needed for performance analysis
- Aggressive Verification Schedule
  - Refer to verification review

Emulator Status

- SQ/SX - 99% Feature Complete
- SP – 97% Feature Complete
  - HW Accuracy in Scalar Unit
- RB/RC – 95% Feature Complete
  - MSAA Resolve
  - Floating Point Depth
- TP/TC – 92% Feature Complete
  - MSAA Resolve
  - Couple of Formats
  - HW Accurate Filter
  - MSAA LOD Correction

R400 Directed Tests

- RB/RC Test Plan
  - ~720 tests
  - ~390 completed
  - 10 / 7 Estimated date of completion (mostly MSAA, TB tests left)
- SQ/SP/SX Test Plan
  - ~1600 Tests
  - ~1400 Completed
  - 9 / 20 Estimated date of completion
- TP / TC Test Plan
  - ~2200 Tests
  - ~400 Completed
  - 10 / 15 Estimated date of completion

R400 Directed Tests (cont)

- MC / MH
  - ~60 Tests Written
  - Test bench coming to completion (GART being added)
  - Heavy Emphasis on Random
- (To do) Chip / GC Specific Testing Plan
  - Surface Synchronization / Events
  - Low Power Modes
  - Interrupts

R400 Random Testing

- Similar to R300 Methodology
  - Bottoms up approach
  - Module Class w/ Initialize, Randomize, Update Methods
- Staffing
  - GC, TP and Infrastructure (George V.)
  - VS, PS (Ken M.)
  - VGT, PA, SC, Primitive (Scott H. / Clay T.)
  - RB (Mark F.)
  - Primlib (Kevin R.)
- Schedule
  - 9/6 – Primlib render state methods implemented
  - 9/13 – Infrastructure in place (base Class, etc…)
  - 10/15 – GC Running

R400 GC Debug

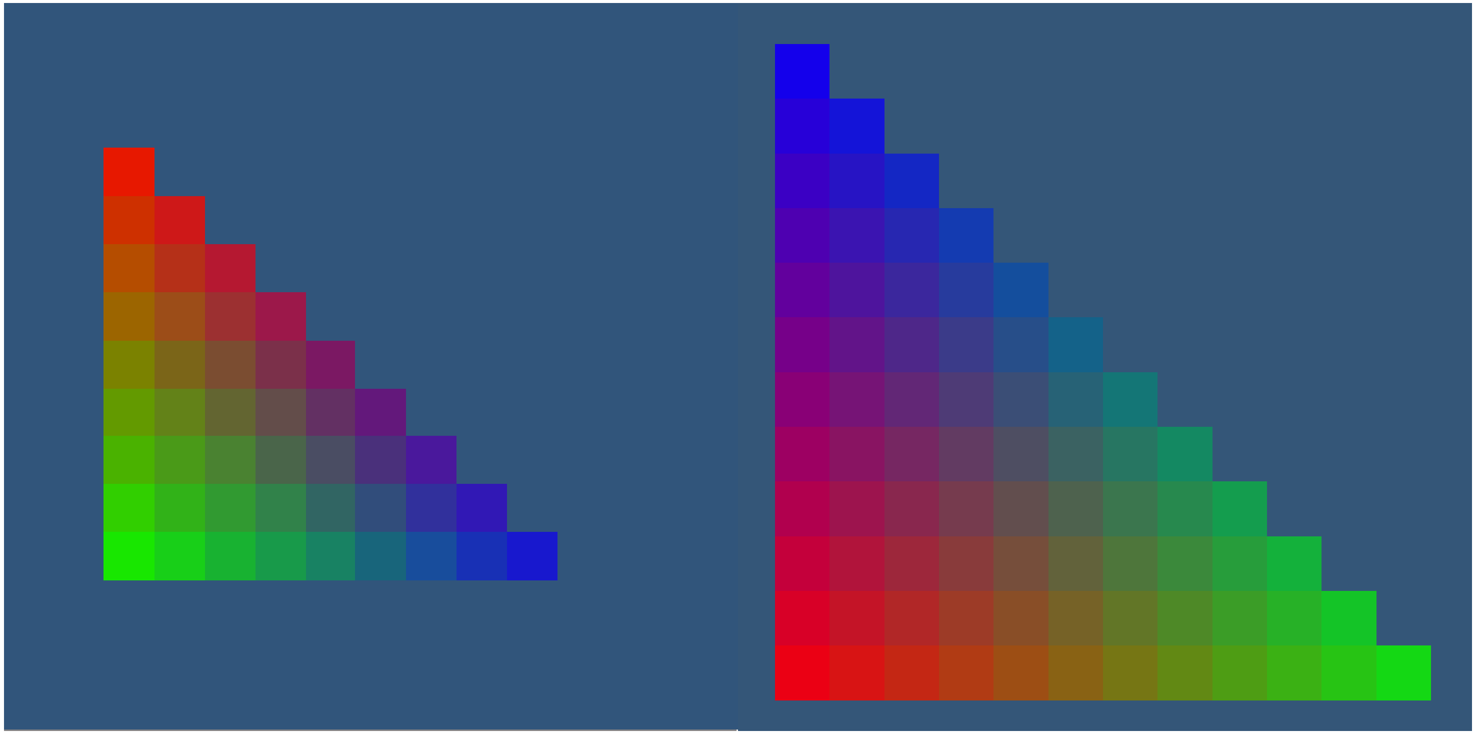
- Status
  - Multiple pixel vectors working through SQ/SP/SX, but not arbitrarily large triangles.
  - Multiple primitives working through SQ/SP/SX, but not arbitrary #.
  - Large triangles working through RB/RC, tile cache bug w/ very large triangles.
  - Debugging cache flush event (needed for pass/fail indication of depth buffer tests).
  - Debugging context done event (needed for multiple contexts).

ATI TECHNOLOGIES INC.

R400 August Program Review Summary

ATI 2066
LG v. ATI
IPR2015-00326

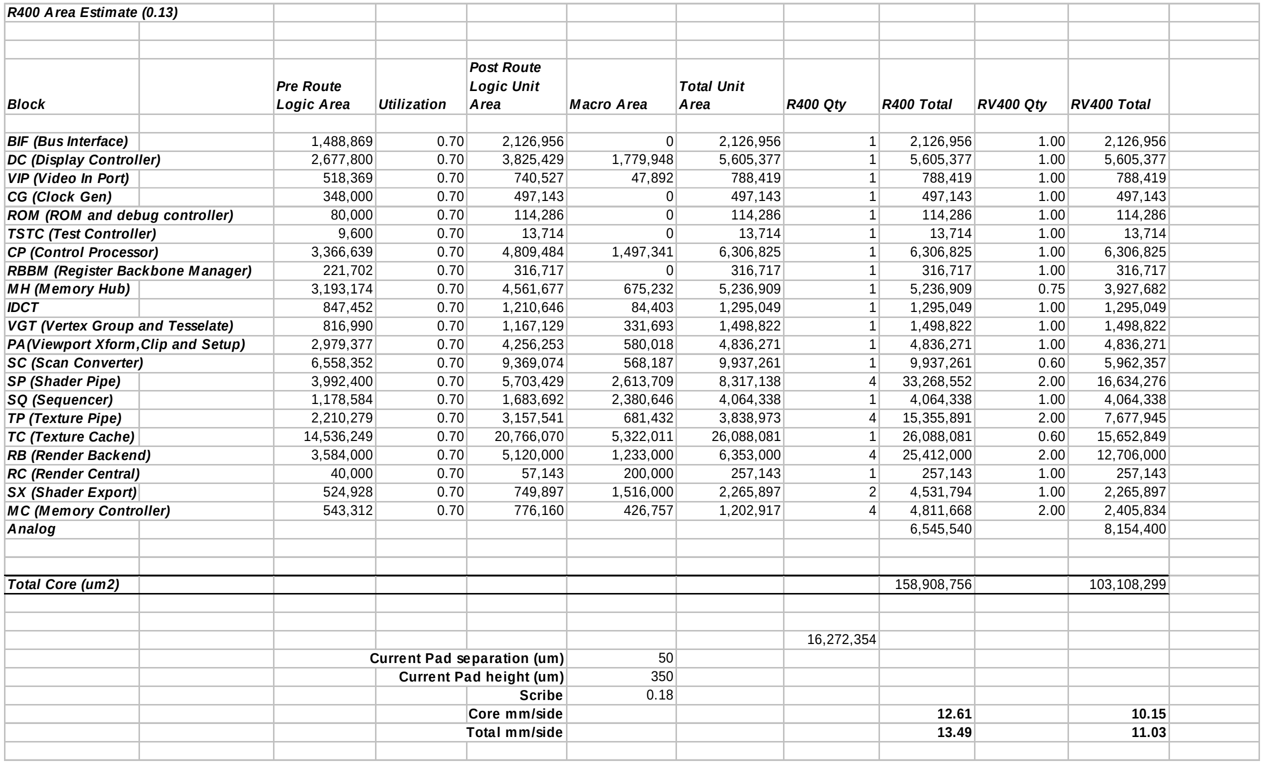
R400 First HW Triangle(s)



Area Estimate

Assumptions

• Routing efficiency	70%
• Pad size (RV350)	80µ x 350µ
• R400 in 0.13µ	
• Core Size	12.6mm
• Total area (pad limited)	~14mm
• RV450 in 0.09µ (estimated)	
• Core Size	8.2mm
• Total Area	9.1mm



R400 Area Estimate (0.13)

Block	Pre Route Logic Area	Utilization	Post Route Logic Unit Area	Macro Area	Total Unit Area	R400 Qty	R400 Total	RV400 Qty	RV400 Total
BIF (Bus Interface)	1,488,869	0.70	2,126,956	0	2,126,956	1	2,126,956	1.00	2,126,956
DC (Display Controller)	2,677,800	0.70	3,825,429	1,779,948	5,605,377	1	5,605,377	1.00	5,605,377
VIP (Video In Port)	518,369	0.70	740,527	47,892	788,419	1	788,419	1.00	788,419
CG (Clock Gen)	348,000	0.70	497,143	0	497,143	1	497,143	1.00	497,143
ROM (ROM and debug controller)	80,000	0.70	114,286	0	114,286	1	114,286	1.00	114,286
TSTC (Test Controller)	9,600	0.70	13,714	0	13,714	1	13,714	1.00	13,714
CP (Control Processor)	3,366,639	0.70	4,809,484	1,497,341	6,306,825	1	6,306,825	1.00	6,306,825
RBBM (Register Backbone Manager)	221,702	0.70	316,717	0	316,717	1	316,717	1.00	316,717
MH (Memory Hub)	3,193,174	0.70	4,561,677	675,232	5,236,909	1	5,236,909	0.75	3,927,682
IDCT	847,452	0.70	1,210,646	84,403	1,295,049	1	1,295,049	1.00	1,295,049
VGT (Vertex Group and Tesselate)	816,990	0.70	1,167,129	331,693	1,498,822	1	1,498,822	1.00	1,498,822
PA(Viewport Xform,Clip and Setup)	2,979,377	0.70	4,256,253	580,018	4,836,271	1	4,836,271	1.00	4,836,271
SC (Scan Converter)	6,558,352	0.70	9,369,074	568,187	9,937,261	1	9,937,261	0.60	5,962,357
SP (Shader Pipe)	3,992,400	0.70	5,703,429	2,613,709	8,317,138	4	33,268,552	2.00	16,634,276
SQ (Sequencer)	1,178,584	0.70	1,683,692	2,380,646	4,064,338	1	4,064,338	1.00	4,064,338
TP (Texture Pipe)	2,210,279	0.70	3,157,541	681,432	3,838,973	4	15,355,891	2.00	7,677,945
TC (Texture Cache)	14,536,249	0.70	20,766,070	5,322,011	26,088,081	1	26,088,081	0.60	15,652,849
RB (Render Backend)	3,584,000	0.70	5,120,000	1,233,000	6,353,000	4	25,412,000	2.00	12,706,000
RC (Render Central)	40,000	0.70	57,143	200,000	257,143	1	257,143	1.00	257,143
SX (Shader Export)	524,928	0.70	749,897	1,516,000	2,265,897	2	4,531,794	1.00	2,265,897
MC (Memory Controller)	543,312	0.70	776,160	426,757	1,202,917	4	4,811,668	2.00	2,405,834
Analog							6,545,540		8,154,400
Total Core (um2)							158,908,756		103,108,299
						16,272,354			
			Current Pad separation (um)	50					
			Current Pad height (um)	350					
			Scribe	0.18					
			Core mm/side				12.61		10.15
			Total mm/side				13.49		11.03

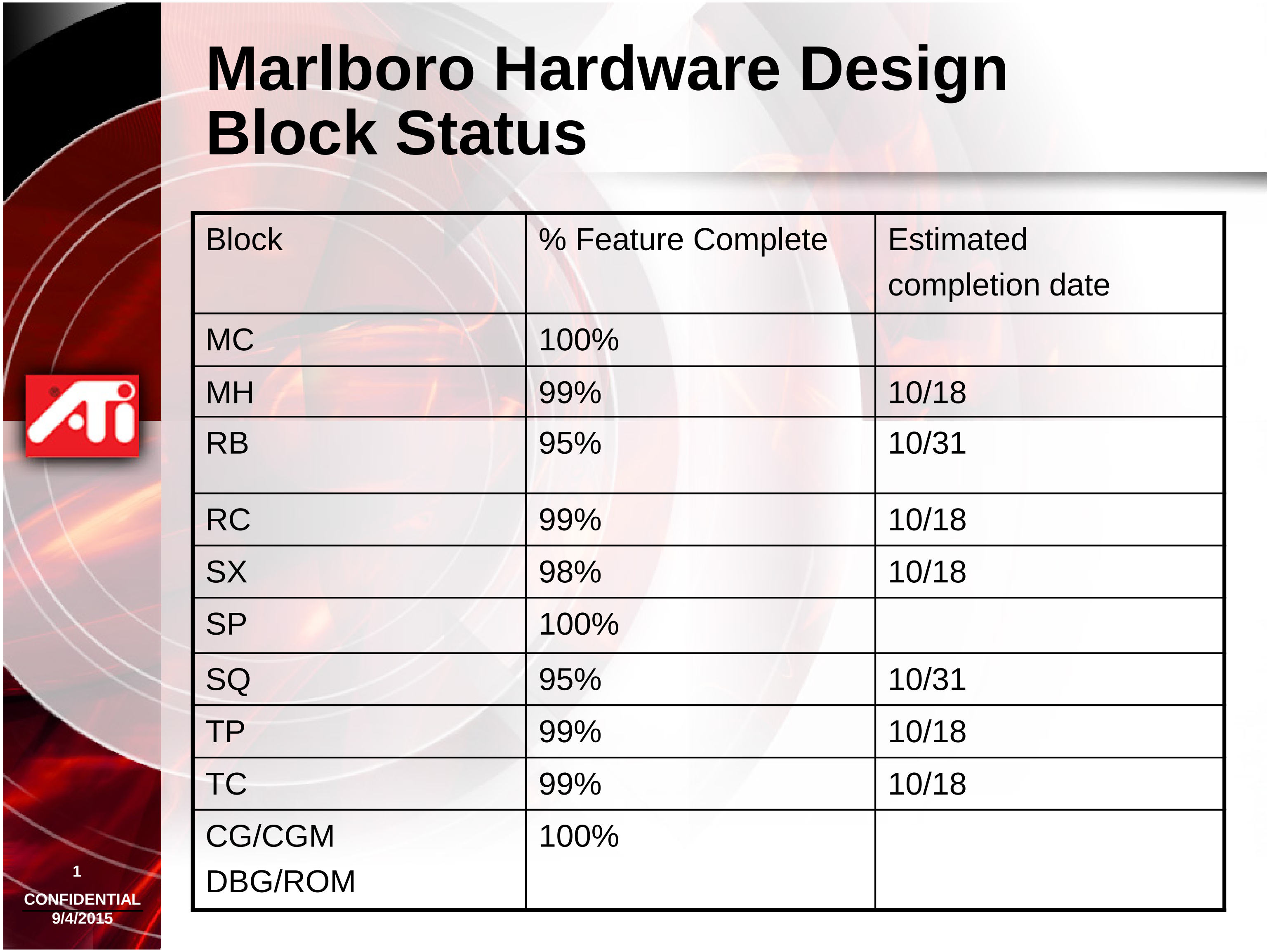
Block estimates

R400 Summary

- R400 Technology
  - Original plan called for 0.13HS+ TSMC process
  - Now using TSMC 0.13 micron LV/LVOD process to lower risk (low K ?)
- R400 IP
  - Leveraging RV350 analog IP (designed in Toronto) as much as possible
- Chip Integration
  - R400 pads to be done by RV350 team – need support for 600MHz DDR
  - Repairable memories via Virage STAR compiler (complete functional compiler delivery late)
- Physical Design
  - ICDG resources worked with initial netlist. SiV resources to continue (going forward)
  - Work out PD deliverables / schedule with Bob Patel
- Overall
  - HW Feature complete / Debug progressing (but slower in some areas than planned)
  - Addressing SQ block debug
  - February tapeout.
  - Continue to work with Bob Patel and team on physical design

R400 Program Schedule

Task	Plan	Actual	Forecast
Significant Architecture Issue Identification Complete	10-15-01	10-10-01	
Emulator Test template Complete	01-18-02	01-18-02	
GC Emulator integration – 1 triangle	02-22-02	02-21-02	
Core Emulator pixel / shader tests run	03-15-02	03-19-02	
Block Testing Begins	04-16-02	05-01-02	
GC/Chip Integration Start	05-17-02	05-15-02	
Simulate 1 Triangle / Emulator ready for SW	06-15-02	07-01-02	
First Syntheses	07-12-02	08-03-03	
Verilog Feature Complete	09-16-02		09-30-02
IKOS Emulation start	10-11-02		10-11-02
Begin early block delivery	11-08-02		11-15-02
IKOS Emulation (w/ Software) begins	11-11-02		11-11-02
RTL Freeze / Final Netlist (Gate level ECO only)	11-15-02		12-15-02
A11 Base Layers Tapeout	01-10-03		02-14-03
A11 Metal Layers Tapeout	01-24-03		02-28-03
First Samples for Engineering			05-09-03
A12 Tapeout			06-14-03
A12 Samples for Engineering			07-12-03
R400 Customer Samples			07-19-03
Volume Ramp			08-19-03
Product Delivery			09-01-03



Marlboro Hardware Design Block Status

Block	% Feature Complete	Estimated completion date
MC	100%	
MH	99%	10/18
RB	95%	10/31
RC	99%	10/18
SX	98%	10/18
SP	100%	
SQ	95%	10/31
TP	99%	10/18
TC	99%	10/18
CG/CGM DBG/ROM	100%	

Marlboro Hardware Design Block Status

- All blocks need to add
  - performance counters
  - power
  - debug registers
  - Star memories
- MC
  - Pad Interface
  - At speed sw generated commands
  - Synthesis target of 600MHz
- MH
  - Register updates in progress
  - Tiled surfaces in system memory support not started

Marlboro Hardware Design Block Status

- RB
  - Depth and multi-sample logic still in design
  - Resolve and memory export logic not yet started
- SQ
  - Control flow in progress
    - (conditionals, loops, jumps, …)
- SP/ SX
  - Memory export

Marlboro Hardware Design Block Status

- TP / TC
  - 512 bit AGP requests
  - degamma with DXTC texels
- CG / CGM / DBG / ROM / IO
  - DFT logic in progress
  - Temperature sensor logic

Block Synthesis

- Update …

Emulator Status

- SQ/SX/SP - 100% Feature Complete
  - Floating point details TBD
- RB/RC – 95% Feature Complete
  - MSAA Resolve
  - Degamma
  - Exporting depth from shader
  - Alpha source to sample mask conversion
- TP/TC – 95% Feature Complete
  - MSAA Resolve
  - MSAA LOD Correction
  - Degamma

R400 Directed Tests

- RB/RC Test Plan
  - ~820 tests in plan
  - ~735 written
  - 10 / 21 Estimated date of completion (mostly MSAA, TB tests left)
- SQ/SP/SX Test Plan
  - ~1600 tests in plan
  - ~1550 written
  - 10 / 15 Estimated date of completion
- TP / TC Test Plan
  - ~5500 tests in plan
  - ~3800 written
  - 10 / 25 Estimated date of completion

R400 Directed Tests (cont)

- MC / MH
  - ~2200 tests in plan
  - ~600 written
- Chip / GC Specific Testing Plan
  - Surface Synchronization / Events
  - Power Management
  - Interrupts
  - AGP / ATI GART Apertures

R400 Random Testing

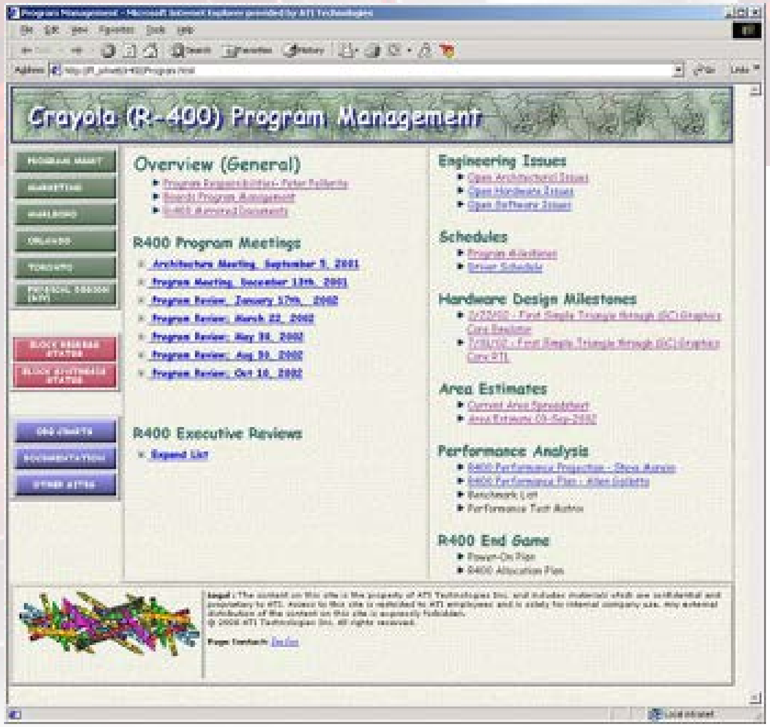
- Similar to R300 Methodology
  - Bottoms up approach
  - Module Class w/ Initialize, Randomize, Update Methods
- Staffing
  - GC, TP and Infrastructure (George V.)
  - VS, PS (Ken M.)
  - VGT, PA, SC, Primitive (Scott H. / Clay T.)
  - RB (Mark F.)
  - Primlib (Kevin R.)
- Schedule
  - 9/6 – Primlib render state methods implemented
  - 9/13 – Infrastructure in place (base Class, etc…)
  - 10/15 – GC Running

R400 Program Review

October 10, 2002

R400 Program Website

- http://fl_orlweb/r400/Program.html



Crayola (R-400) Program Management

Overview (General)

- Program Responsibilities- Peter Roberts
- Boards Program Management
- R400 Approved Documents

R400 Program Meetings

- Architecture Meeting, September 5, 2001
- Program Meeting, December 13th, 2001
- Program Review, January 17th, 2002
- Program Review, March 22, 2002
- Program Review, May 30, 2002
- Program Review, Aug 30, 2002
- Program Review, Oct 10, 2002

R400 Executive Reviews

- Expand List

Engineering Issues

- Open Architectural Issues
- Open Hardware Issues
- Open Software Issues

Schedules

- Program Milestones
- Driver Schedule

Hardware Design Milestones

- 2/22/02 - First Simple Triangle through GC Graphics Core Emulator
- 7/30/02 - First Simple Triangle through GC Graphics Core RTL

Area Estimates

- Current Area Spreadsheet
- Area Estimate 05-Sep-2002

Performance Analysis

- R400 Performance Projection - Steve Morein
- R400 Performance Plan - Allen Gotlieb
- Benchmark List
- Performance Test Matrix

R400 End Game

- Power-On Plan
- R400 Allocation Plan

Legal : The content on this site is the property of ATI Technologies Inc. and includes materials which are confidential and proprietary to ATI. Access to this site is restricted to ATI employees and is solely for internal company use. Any external distribution of the content on this site is expressly forbidden.
© 2002 ATI Technologies Inc. All rights reserved.

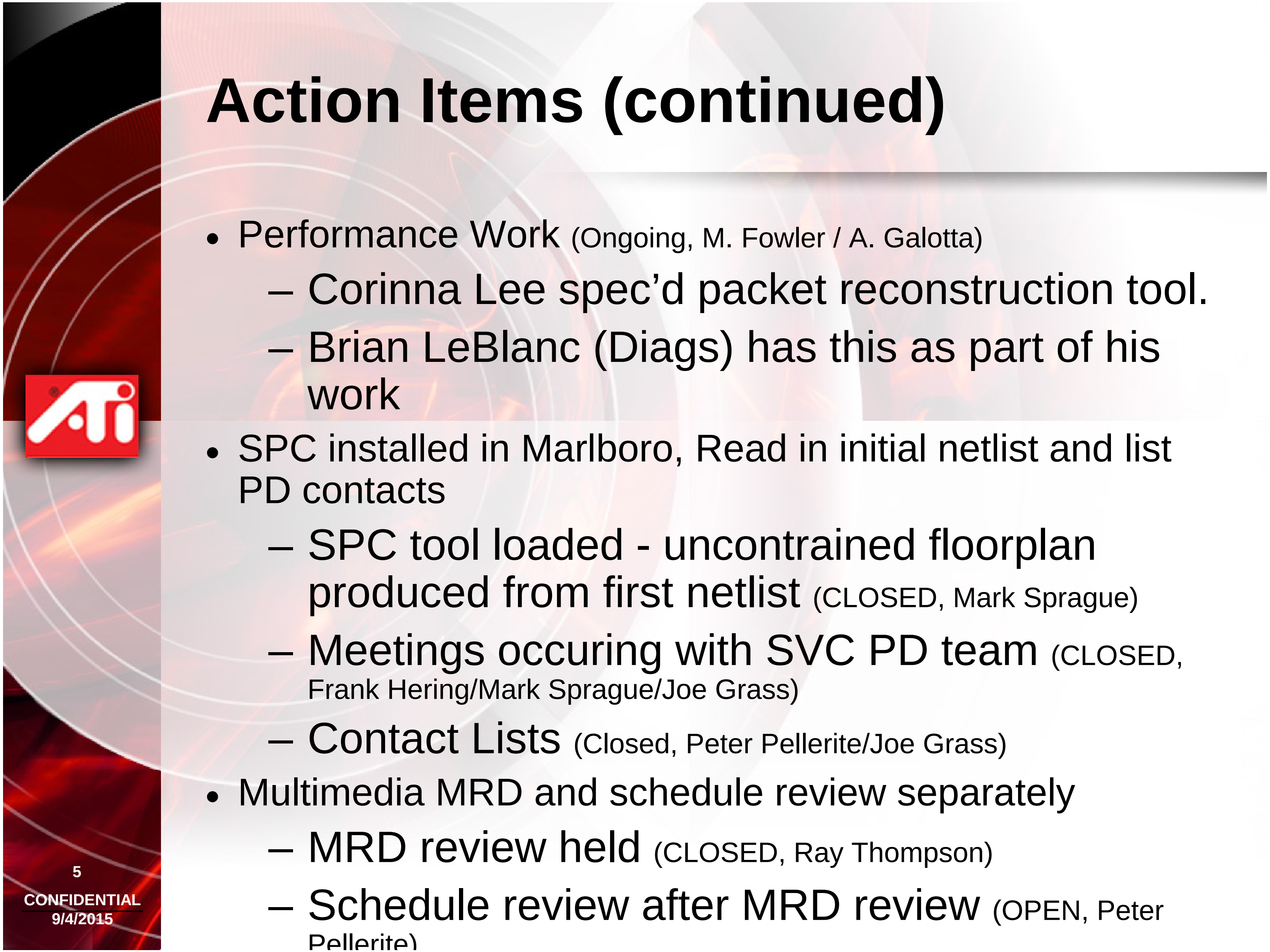
Page Contact: [illegible]

Action Item Status (last review)

- 64b render targets (Closed, Peter Pellerite / Jay Wilkinson)
  - Savings in area vs. schedule impact
  - Feature was already coded, more impact to remove
  - RB team is concentrating on debug; area savings next
- MS to document Direct X decisions (OPEN, Bob Feldstein)
  - (For example, export to memory as an extension)
  - Bob Feldstein is having ongoing discussions
- TV/CRT DAC usage in R400 issue (Closed, Ray Thompson)
  - Will use one TV DAC one CRT DAC in R400
- Requirement for support fof 2 CRTs and TV at the same time (Closed, Ray Thompson)
  - Yes, feature required
  - Analog MUX will be external
  - Additional pin added to R400 for control

Action Items (continued)

- Chip ID definition
  - Chip IDs reserved (Closed, Peter Pellerite)
  - "Personality" bit definition 10/10/02 (Open, Frank Hering)
- Virage SMS (Star) memory delay (Open, Frank Hering)
  - Meeting with Virages to discuss
  - HD memories for R400 to use 2.8micron bit cell
  - STAR system meeting 10/11/02
- Coordination of multi-site IKOS and full chip tests (Closed)
  - Frank Hering is coordinating IKOS work with Colin Stewart & Ron White
  - Each site is running sets of full chip tests, Christeen Gray is coordinating system validation
- Clock Skew (Open, Hering / Sprague)
  - Coordinate with SVC, PD experiments needed
  - Using 300pS until a better number is available



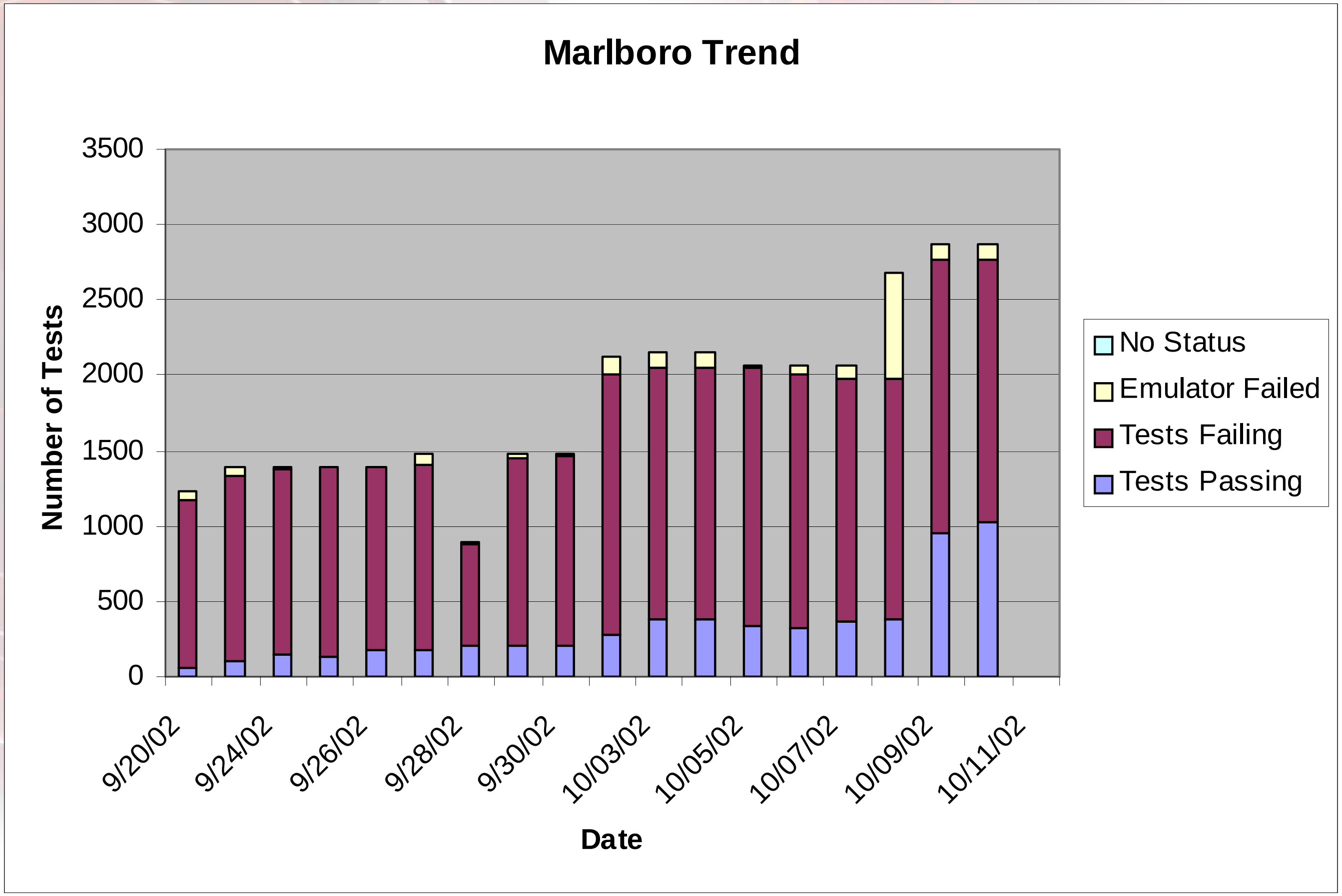
Action Items (continued)

- Performance Work (Ongoing, M. Fowler / A. Galotta)
  - Corinna Lee spec’d packet reconstruction tool.
  - Brian LeBlanc (Diags) has this as part of his work
- SPC installed in Marlboro, Read in initial netlist and list PD contacts
  - SPC tool loaded - uncontrained floorplan produced from first netlist (CLOSED, Mark Sprague)
  - Meetings occuring with SVC PD team (CLOSED, Frank Hering/Mark Sprague/Joe Grass)
  - Contact Lists (Closed, Peter Pellerite/Joe Grass)
- Multimedia MRD and schedule review separately
  - MRD review held (CLOSED, Ray Thompson)
  - Schedule review after MRD review (OPEN, Peter Pellerite)

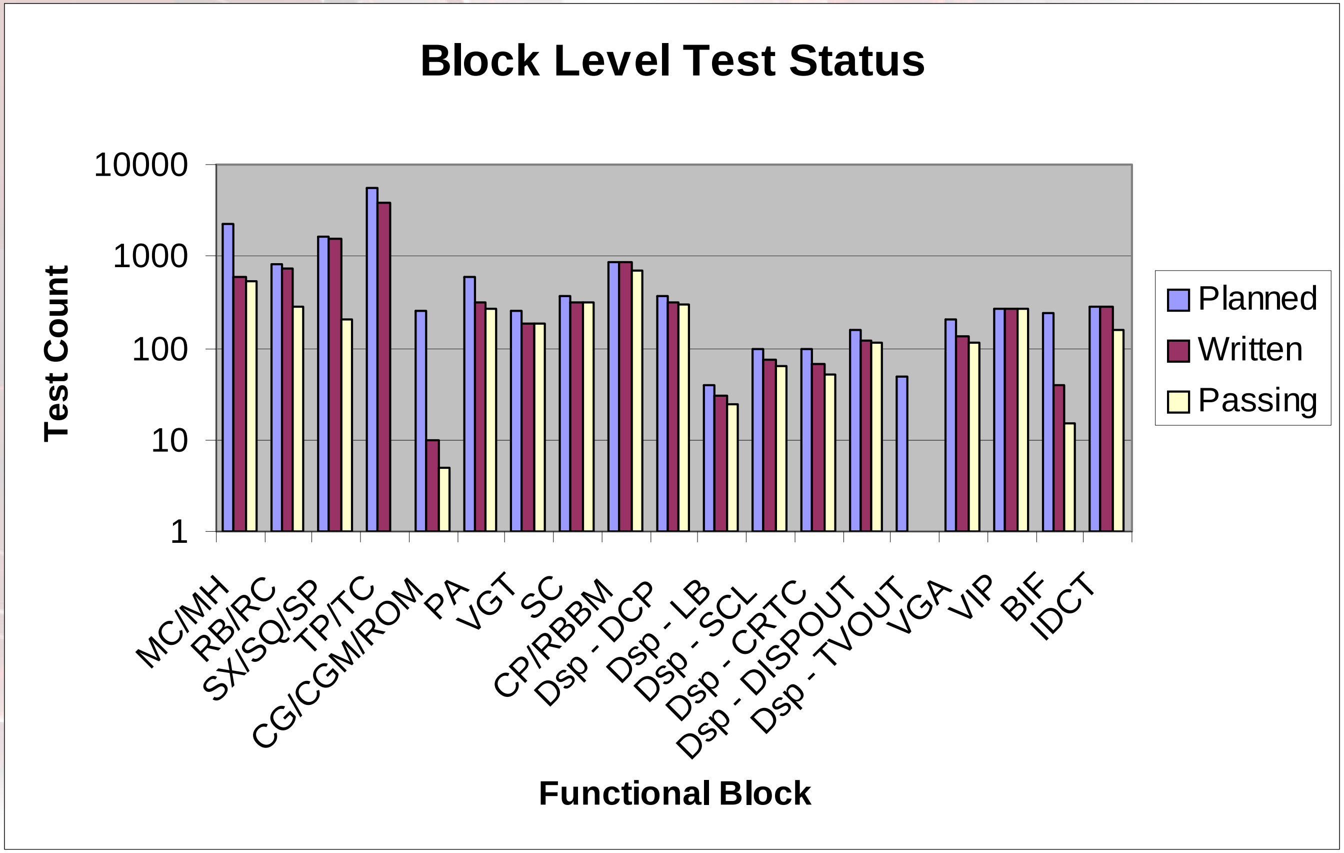
Block Logic Design and Timing

Block	Percent Feature Complete	Forecast Feature Complete	Emulator Complete	Directed Tests Written	Synthesis W/C		TNS (nS)
					SCLK (2.5nS)	MCLK (2nS) PIXCLK (2.5nS) BCLK (14.1nS)	
MC/MH*	100% / 99%	Done, 10/18	N/A	27%	MC:2.5; MH: 3.30	MC - MCLK: 2.00	0 / 14491
RB*/RC	95% / 99%	10/31	95%	90%	RB: 6.3; RC 2.5		6211 / 0
SX*/SQ	98% / 95%	10/18, 10/31	100%	97%	SX: 2.5; SQ: 2.51		0 / 0
SP	100%	Done	100%		SP: 2.5		0
TP/TC*	99% / 99%	10/18	95%	69%	TP: 2.5; TC: 7.5		0 / 0
CG/CGM/ROM	100%	Done	N/A	4%	CG: 2.5; CGM:2.5; DBG:2.5		0
PA	100%	Done	100%	36%	PA: 2.79		109
VGT	100%	Done	100%	70%	VGT: 2.50		0
SC*	>95%	10/15	>97%	68%	SC:3.4		35
RBBM	95%	10/31*	N/A	90%	RBBM: 2.5		0
CP	95%	10/31*	95%		CP: 3.1		408
Display - DCP	100%	Done	95%	86%	Display: 3.93	PIXCLK: 2.66	3608
Display - LB	100%	Done	100%	75%	2.07		
Display - SCL	100%	Done	90%	75%	4.58		
Display - CRTC	100%	Done	100%	71%	2.19		
Display - DISPOUT	100%	Done	90%	74%	2.16		
Display - TVOUT	100%	Done	40%	0%		21.81(TVCLK:15ns)	
VGA	100%	Done	99%	65%	4.27		
VIP	100%	Done	100%	100%	2.94		
BIF	95%	10/11	90%	17%	BIF 2.96	BCLK: 14nS; AGP:3.5nS	242
IDCT	100%	Done	97%	97%	IDCT: 2.33		

Block Testing



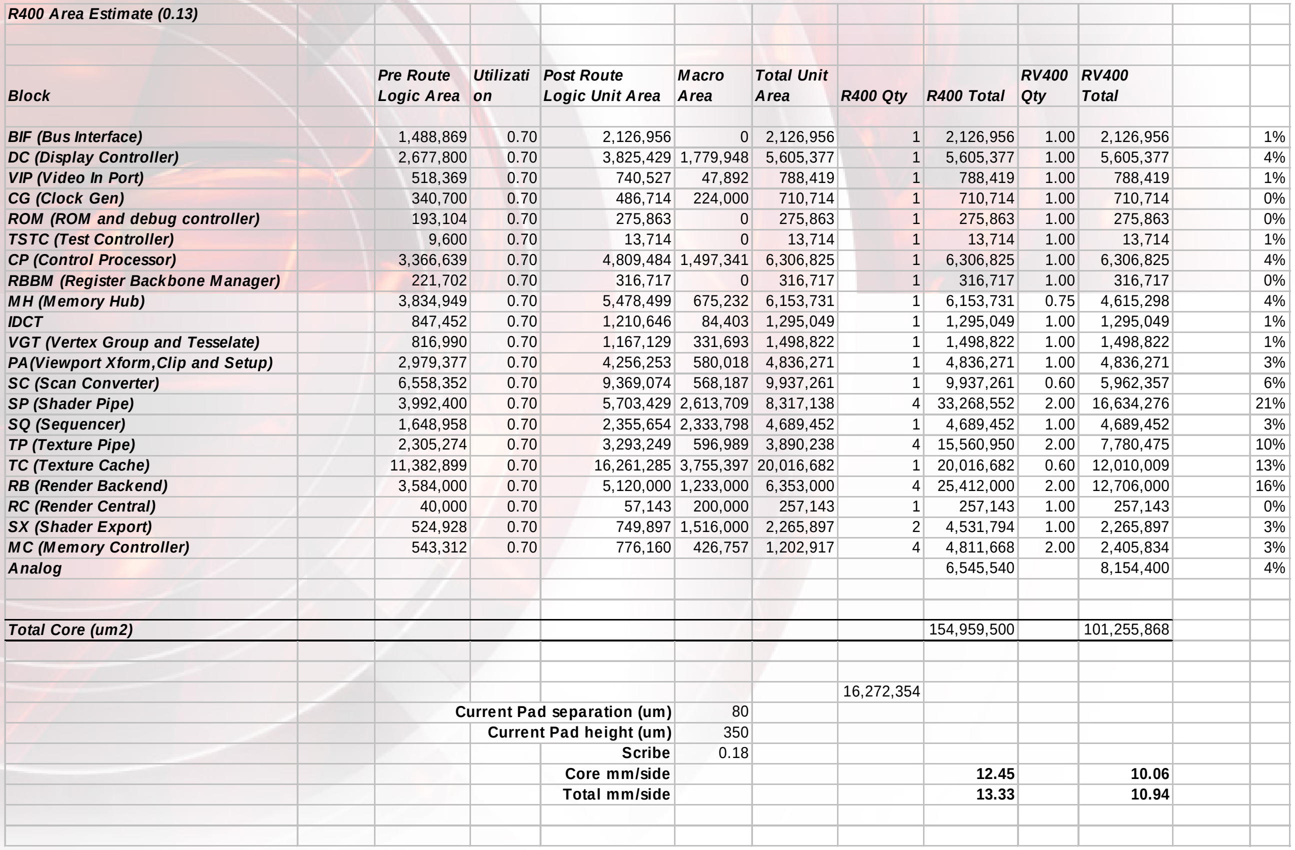
Block Testing



Area Summary

• Routing efficiency	70%
• Pad size (RV350)	80µ x 350µ
• R400 in 0.13µ	
– Core Size	12.45mm
– Total area (pad limited ?)	~14mm
• RV450 in 0.09µ (estimated)	
– Core Size	8.1mm
– Total Area	9.1mm

R400 Area Summary

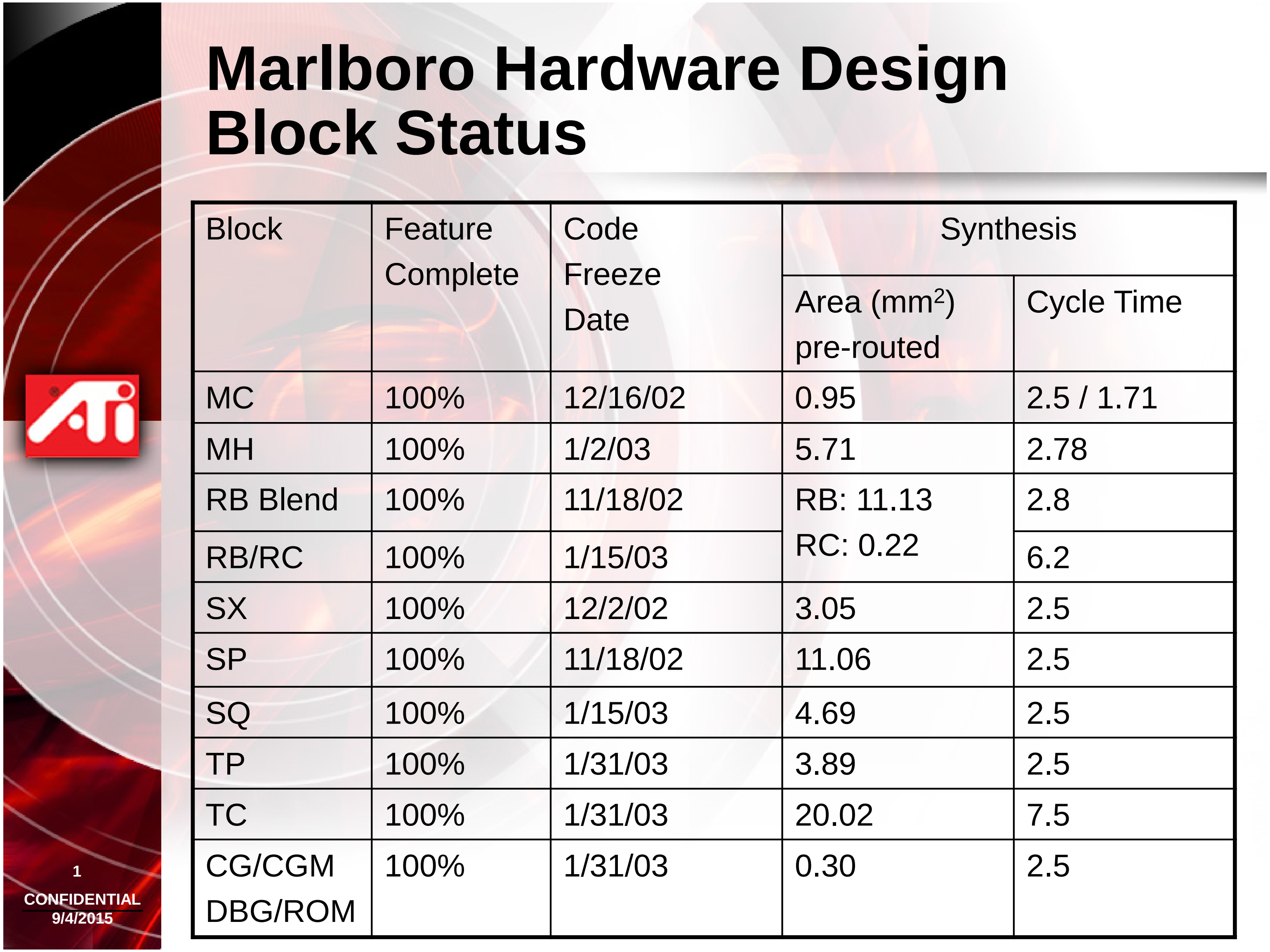


R400 Area Estimate (0.13)											
Block	Pre Route Logic Area	Utilizati on	Post Route Logic Unit Area	Macro Area	Total Unit Area	R400 Qty	R400 Total	RV400 Qty	RV400 Total		
BIF (Bus Interface)	1,488,869	0.70	2,126,956	0	2,126,956	1	2,126,956	1.00	2,126,956		1%
DC (Display Controller)	2,677,800	0.70	3,825,429	1,779,948	5,605,377	1	5,605,377	1.00	5,605,377		4%
VIP (Video In Port)	518,369	0.70	740,527	47,892	788,419	1	788,419	1.00	788,419		1%
CG (Clock Gen)	340,700	0.70	486,714	224,000	710,714	1	710,714	1.00	710,714		0%
ROM (ROM and debug controller)	193,104	0.70	275,863	0	275,863	1	275,863	1.00	275,863		0%
TSTC (Test Controller)	9,600	0.70	13,714	0	13,714	1	13,714	1.00	13,714		1%
CP (Control Processor)	3,366,639	0.70	4,809,484	1,497,341	6,306,825	1	6,306,825	1.00	6,306,825		4%
RBBM (Register Backbone Manager)	221,702	0.70	316,717	0	316,717	1	316,717	1.00	316,717		0%
MH (Memory Hub)	3,834,949	0.70	5,478,499	675,232	6,153,731	1	6,153,731	0.75	4,615,298		4%
IDCT	847,452	0.70	1,210,646	84,403	1,295,049	1	1,295,049	1.00	1,295,049		1%
VGT (Vertex Group and Tesselate)	816,990	0.70	1,167,129	331,693	1,498,822	1	1,498,822	1.00	1,498,822		1%
PA(Viewport Xform,Clip and Setup)	2,979,377	0.70	4,256,253	580,018	4,836,271	1	4,836,271	1.00	4,836,271		3%
SC (Scan Converter)	6,558,352	0.70	9,369,074	568,187	9,937,261	1	9,937,261	0.60	5,962,357		6%
SP (Shader Pipe)	3,992,400	0.70	5,703,429	2,613,709	8,317,138	4	33,268,552	2.00	16,634,276		21%
SQ (Sequencer)	1,648,958	0.70	2,355,654	2,333,798	4,689,452	1	4,689,452	1.00	4,689,452		3%
TP (Texture Pipe)	2,305,274	0.70	3,293,249	596,989	3,890,238	4	15,560,950	2.00	7,780,475		10%
TC (Texture Cache)	11,382,899	0.70	16,261,285	3,755,397	20,016,682	1	20,016,682	0.60	12,010,009		13%
RB (Render Backend)	3,584,000	0.70	5,120,000	1,233,000	6,353,000	4	25,412,000	2.00	12,706,000		16%
RC (Render Central)	40,000	0.70	57,143	200,000	257,143	1	257,143	1.00	257,143		0%
SX (Shader Export)	524,928	0.70	749,897	1,516,000	2,265,897	2	4,531,794	1.00	2,265,897		3%
MC (Memory Controller)	543,312	0.70	776,160	426,757	1,202,917	4	4,811,668	2.00	2,405,834		3%
Analog							6,545,540		8,154,400		4%
Total Core (um2)							154,959,500		101,255,868		
						16,272,354					
			Current Pad separation (um)	80							
			Current Pad height (um)	350							
			Scribe	0.18							
			Core mm/side				12.45		10.06		
			Total mm/side				13.33		10.94		

R400 Risks / Issues Summary

- R400 Technology
  - Proceeding with TSMC 0.13 Low-K; need to continue to monitor status
- Blocks Design & Chip Integration
  - Validation / debug effort
  - Virage STAR compiler - functional compiler delivery late
    - Impacted completeness netlist one; release second netlist with STAR
  - Chip level simulation environment stabilizing - debug of non-GC blocks
- Physical Design
  - Logic Design netlist deliveries, constraint file
  - Early block delivery
- Software
  - BIOS is new (re-architected)
  - Compiler progress / bug fixes
- Overall
  - Tapeout target impact

CONFIDENTIAL
9/4/2015



Marlboro Hardware Design Block Status

Block	Feature Complete	Code Freeze Date	Synthesis	
			Area (mm^2) pre-routed	Cycle Time
MC	100%	12/16/02	0.95	2.5 / 1.71
MH	100%	1/2/03	5.71	2.78
RB Blend	100%	11/18/02	RB: 11.13 RC: 0.22	2.8
RB/RC	100%	1/15/03		6.2
SX	100%	12/2/02	3.05	2.5
SP	100%	11/18/02	11.06	2.5
SQ	100%	1/15/03	4.69	2.5
TP	100%	1/31/03	3.89	2.5
TC	100%	1/31/03	20.02	7.5
CG/CGM DBG/ROM	100%	1/31/03	0.30	2.5

1

CONFIDENTIAL
9/4/2015

Marlboro Hardware Design Block Status – Outstanding Issues

- All blocks need to add
  - performance counters
  - power management
  - CRC checkers and other test specific logic
- MC
  - Pad Interface
- MH
  - Rotate and surface synchronization limited testing
- RB
  - Depth & multi-sample logic in debug
  - Resolve and memory export logic limited testing
  - Area optimization

Marlboro Hardware Design Block Status – Outstanding Issues

- SQ
  - working on optimizations for performance
- SP/ SX
  - No outstanding issues
- TP / TC
  - optimize for 512 bit AGP requests
  - area optimizations / cycle times
- CG / CGM / DBG / ROM / IO
  - IO pad – MC interface
  - Temperature sensor logic contracted to Toronto

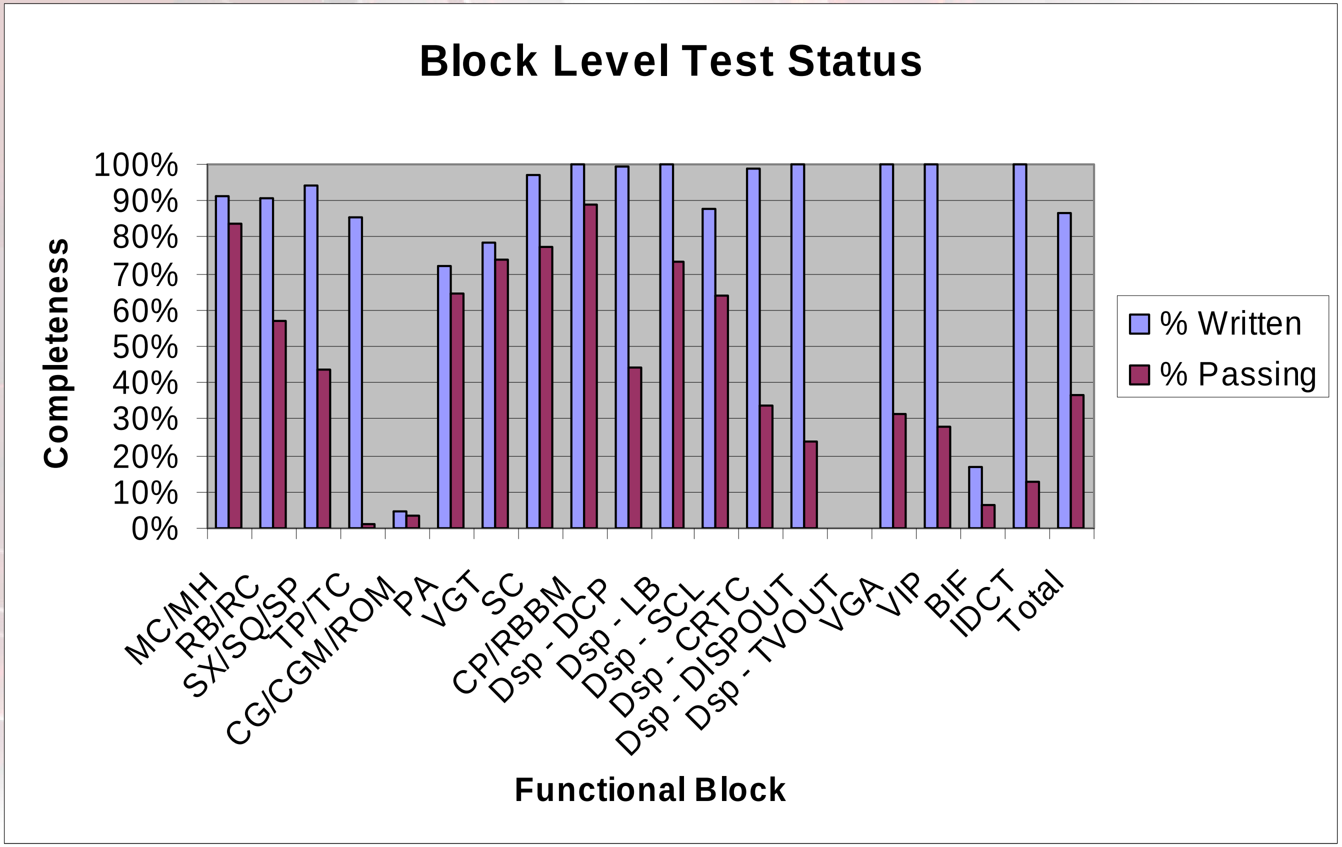
R400 Program Review

November 12, 2002

Block Summary

Block	Percent Feature Complete	Forecast Feature Complete	Emulator Complete	Directed Tests Written	Synthesis W/C		TNS (nS)
					SCLK (2.5nS)	MCLK (1.67nS), PIXCLK (2.5nS), BCLK (14nS), AGP(3.5nS), TVCLK(10nS)	
MC/MH	100%	Done	N/A	91%	MC:2.5; MH: 2.78	MC - MCLK: 1.71	0 / 1295
RB/RC	100%	Done	99%	91%	RB: 6.2; RC 2.5		12155 / 0
SX/SQ	100%	Done	100%	94%	SX: 2.5; SQ: 2.51		0 / 0
SP	100%	Done	100%		SP: 2.5		0
TP/TC	100%	Done	99%	85%	TP: 2.5; TC: 7.5		0 / ?
CG/CGM/ROM	100%	Done	N/A	5%	CG: 2.5; CGM:2.5; DBG:2.5		0
PA	100%	Done	100%	72%	PA: 2.79		142
VGT	100%	Done	100%	79%	VGT: 2.5		0
SC	100%	Done	100%	97%	SC: 2.88		1150
RBBM	100%	Done	95%	100%	RBBM: 2.25		0
CP	100%	Done	95%		CP: 2.83		408
Display - DCP	100%	Done	95%	99%	Display: 4.57	PIXCLK: 2.66	3669
Display - LB	100%	Done	100%	100%			
Display - SCL	100%	Done	90%	88%			
Display - CRTC	100%	Done	100%	99%			
Display - DISPOUT	100%	Done	90%	100%			
Display - TVOUT	100%	Done	40%	6%		TVCLK:10.4	
VGA	100%	Done	99%	100%			
VIP	100%	Done	100%	100%			
BIF	95%	?	90%	17%	BIF 2.5	BCLK: 14.53nS; AGP:3.49nS	242
IDCT	100%	Done	97%	100%	IDCT: 2.5		

Block Testing



Area Summary

- Routing efficiency 70%
- Pad size (RV350) 50µ x 350µ
- R400 in 0.13µ
  - Core Size 13.69mm
  - Total area (pad limited ?) 14.57mm

R400 Area Summary

R400 Area Estimate (0.13)

Block	*Pre Route Logic Area*	*Utilization*	*Post Route Logic Unit Area*	*Macro Area*	*Total Unit Area*	*R400 Qty*	*R400 Total*	*RV400 Qty*	*RV400 Total*
BIF (Bus Interface)	1,481,627	0.70	2,116,609	0	2,116,609	1	2,116,609	1.00	2,116,609
DC (Display Controller)	5,349,478	0.70	7,642,111	1,754,490	9,396,601	1	9,396,601	1.00	9,396,601
VIP (Video In Port)		0.70				1	0	1.00	0
CG (Clock Gen)	123,883	0.70	176,976	0	176,976	1	176,976	1.00	176,976
ROM (ROM and debug controller)	73,107	0.70	104,439	0	104,439	1	104,439	1.00	104,439
TSTC (Test Controller)	9,600	0.70	13,714	0	13,714	1	13,714	1.00	13,714
CP (Control Processor)	2,795,951	0.70	3,994,215	1,553,082	5,547,297	1	5,547,297	1.00	5,547,297
RBBM (Register Backbone Manager)	103,059	0.70	147,228	91,452	238,680	1	238,680	1.00	238,680
MH (Memory Hub)	3,934,344	0.70	5,620,492	908,915	6,529,407	1	6,529,407	0.75	4,897,055
IDCT	1,200,396	0.70	1,714,852	0	1,714,852	1	1,714,852	1.00	1,714,852
VGT (Vertex Group and Tesselate)	756,847	0.70	1,081,210	348,919	1,430,128	1	1,430,128	1.00	1,430,128
PA(Viewport Xform,Clip and Setup)	2,783,577	0.70	3,976,538	755,252	4,731,790	1	4,731,790	1.00	4,731,790
SC (Scan Converter)	7,203,312	0.70	10,290,446	634,693	10,925,139	1	10,925,139	0.60	6,555,083
SP (Shader Pipe)	6,054,940	0.70	8,649,915	2,413,711	11,063,625	4	44,254,501	2.00	22,127,251
SQ (Sequencer)	1,648,958	0.70	2,355,654	2,333,798	4,689,452	1	4,689,452	1.00	4,689,452
TP (Texture Pipe)	2,305,274	0.70	3,293,249	596,989	3,890,238	4	15,560,950	2.00	7,780,475
TC (Texture Cache)	11,382,899	0.70	16,261,285	3,755,397	20,016,682	1	20,016,682	0.60	12,010,009
RB (Render Backend)	6,971,479	0.70	9,959,256	1,167,019	11,126,275	4	44,505,101	2.00	22,252,551
RC (Render Central)	151,388	0.70	216,268	0	216,268	1	216,268	1.00	216,268
SX (Shader Export)	702,959	0.70	1,004,228	2,048,405	3,052,633	2	6,105,266	1.00	3,052,633
MC (Memory Controller)	351,689	0.70	502,413	445,311	947,725	4	3,790,898	2.00	1,895,449
Analog							5,456,740		7,518,580
Total Core (um2)							187,521,491		118,465,893
						17,087,057			
			Current Pad separation (um)	50					
			Current Pad height (um)	350					
			Scribe	0.18					
			Core mm/side				**13.69**		**10.88**
			Total mm/side				**14.57**		**11.76**

5

CONFIDENTIAL
9/4/2015

R400 Risks / Issues Summary

- **R400 Technology**
  - Proceeding with TSMC 0.13 Low-K; need to continue to monitor status
- **Hardware Design & Chip Integration**
  - Validation / debug effort is the long pole
  - Virage STAR compiler
    - Continue to work through implementation issues
  - Test strategy in definition
  - Chip level validation
  - IKOS for Display
- **Physical Design**
  - Working through initial floorplanning over next couple weeks
  - Mature blocks from each site to work through flow
  - Feedback and incorporation of learnings into process
- **Software**
  - Work through staggered netlist delivery and IKOS funtionality
  - Delay for new packet verification
  - Compiler
- **Performance**

High Level Schedule

Task	Plan	Last	Current
Emulator Test template Complete	01-18-02	01-18-02	Complete
GC Emulator integration – 1 triangle	02-22-02	02-21-02	Complete
Core Emulator pixel / shader tests run	03-15-02	03-19-02	Complete
Block Testing Begins	04-16-02	05-01-02	Complete
GC/Chip Integration Start	05-17-02	05-15-02	Complete
Simulate 1 Triangle / Emulator ready for SW	06-15-02	07-01-02	Complete
First Syntheses (n-1)	07-12-02	08-03-03	Complete
Verilog Feature Complete	09-16-02	10-31-02	Complete
Second Synthesis (n)	09-23-02	10-04-02	Complete
IKOS Emulation start	10-11-02	11-15-02	11-22-02
Third Synthesis	11-01-02	11-15-02	11-15-02
Early block delivery	11-08-02	11-18-02	11-18-02
IKOS Emulation (w/ Software) begins	11-11-02	12-16-02	12-16-02
RTL Freeze / Final Netlist (Gate level ECO only)	11-15-02	01-31-03	01-31-03
A11 Base Layers Tapeout	01-10-03	03-28-03	03-28-03
A11 Metal Layers Tapeout	01-24-03	04-11-03	04-11-03
First Samples for Engineering		06-25-03	06-25-03
A12 Tapeout		07-31-03	07-31-03
A12 Samples for Engineering		09-04-03	09-04-03
R400 Customer Samples		09-11-03	09-11-03
Product Delivery		10-04-03	10-04-03

```
Change 11107 on 2001/12/03 by pmitchel@pmitchel_r400_win_marlboro

        mv block dirs to gfx

Change 10478 on 2001/11/21 by askende@andi_r400

        further update of the I/O definition

Change 9918 on 2001/11/14 by askende@andi_r400

        first time check-in

Change 8480 on 2001/10/25 by askende@andi_r400

        inserted into source control by Andi S.

Change 6887 on 2001/09/25 by askende@andi_r400_devel

        more changes

Change 6810 on 2001/09/21 by askende@andi_r400_devel

        newly added files

Change 5440 on 2001/08/16 by askende@andi_r400_devel

        adding source code into source control

Change 5002 on 2001/08/02 by pmitchel@pmitchel_test_client

        directory creation
```